Universidad de Valladolid

Escuela de Ingenierías Industriales

Departamento de Ingeniería de Sistemas y Automática

TESIS DOCTORAL:

AGENT-BASED MODELLING AND SWARM INTELLIGENCE IN SYSTEMS ENGINEERING

Presentada por María Pereda García para optar al grado de doctora por la Universidad de Valladolid

Dirigida por:
Dr. Jesús M. Zamarreño Cosme

Valladolid, Noviembre de 2013.

Universidad de Valladolid

School of Industrial Engineering

Department of System Engineering and Automatic Control

PHD THESIS:

AGENT-BASED MODELLING AND SWARM INTELLIGENCE IN SYSTEMS ENGINEERING

A Thesis submitted by María Pereda García for the degree of Doctor of Philosophy in the University of Valladolid

Supervised by:
Jesús M. Zamarreño Cosme, PhD

Valladolid, November 2013.

Reach for the sky
cause tomorrow may never come.

Acknowledgements

In this lines I want to remember and thank to all the people that has contributed to this thesis in some way.

This thesis would not have been possible without the help of my advisor, Jesús M. Zamarreño. I want to thank him for his tremendous dedication, his motivational skills and his effort spent in helping me with valuable comments and advices and guiding me during these four years.

I also want to thank to my partners of the System Engineering and Automatic Control department, and specially to my PhD colleagues for the good times in meetings and conferences. I wish them all the best.

I would also like to express my deepest gratitude to Marta Ginovart, for her advices and valuable contributions, and also for her hospitality during my stay in Barcelona.

I also gratefully acknowledge the research grants program *Ayuda para la Formación de Personal Investigador* from the University of Valladolid and the research projects DPI2009-14410-C02-02 and DPI 2012-39381-C02-02.

I also want to remember and give thanks to my friend Josema, for the advices and reading material he suggested me.

I cannot forget my dear friends, I want to thank them the support, the chats, the good moments together... They make life happier.

I want to thank to my love, Manu, for everything, for being a reason to live for.

And thank you to my family: mom, dad and brother, for your unconditional support, and for making home a sweet home.

Thank you,

María

Summary

A smooth sea never made a skilled mariner.
English Proverb

Agent-based modelling is a modelling methodology that has revolutionized the field of modelling and simulation of complex systems, being a reliable alternative to traditional lumped parameter models, phenomenological or empirical. This technique aims to obtain models of systems representing the constituents of such systems. It is a bottom-up approximation where the properties of the systems are the result (emerge) from the aggregation of the properties of the system elements and the relations between these elements, the *agents*. This technique has been successfully applied in various fields: economics, social sciences, medicine, biology, etc.

The Process systems engineering field is responsible for modelling complex systems whose dynamic nature implies that this process is not a simple task, resulting in nonlinear models, models in which we must take into account the presence of disturbances, past states dependencies, etc. This modelling technique (agent-based modelling) is pretty adequate to modelling complex systems. The processes of wastewater treatment have a complex dynamic due to variations in the flow rate and composition of the influent of the wastewater treatment plants and the large number of processes and microorganisms involved, which makes the control of these plants rather complicated. This makes the activated sludge process an ideal system to apply agent-based modelling. The developed model has allowed us to confirm the decisive influence the composition and flow of the influent to a wastewater treatment plant has in its dynamics. The model, once validated with respect to a real plant, could be incorporated, for example, into a model based predictive control scheme, where predictions given by the model could be employed to obtain the best control actions to be applied to the system (with respect to certain optimization objectives). Examples of these objectives are: to avoid undesirable sewage flows outside the main sewers, to minimize the control energy (the energy cost of the movements of regulation gates), to maximize sewage treatment, to reduce the volume in the tanks

in anticipation for upcoming rainstorms, etc [93]. Control of the activated sludge process involves keeping the substrate concentration below a certain level established by law, maintaining enough concentration of biomass to allow degradation of the substrate, and maintaining enough oxygen to allow these reactions but minimizing it since it implies a cost. The main difficulties in controlling this process are the lack of measurable information online (estimators would be needed) and the complexity of the process (flow and load variations, weather influence) [88].

In this thesis we have applied agent-based modelling to the activated sludge process, which takes place in the process of sewage treatment. We have studied the methodology, the different options of implementation, developed a model (a simplified version of the process with one type of bacteria and one type of substrate), and developed a notation to complement the current representation standard.

In addition, in this thesis we have worked with the *Swarm Intelligence* paradigm, that is, patterns and macroscopic behaviours that result from the collaboration of small entities with simple rules and without a leadership figure, for example, ant colonies, flock of birds, etc ... Based on this *Swarm* philosophy, somehow related with the agent-based modelling, some techniques arise. In this thesis we have worked with optimization algorithms, testing the *Particle Swarm Optimization* and designing our own algorithm, the *Hiker*, which shows promising results, as seen in this document.

Usually, in the systems engineering and automatic control field, it is needed that algorithms and models are executed as fast as possible to be used in real time. Motivated by the need of reducing this computational time, in this thesis we have worked with code parallelization techniques, and in particular, parallel code to run on graphics cards. These techniques have been used both in agent-based modelling and in swarm optimization algorithms.

In summary, we have applied agent-based modelling to the activated sludge process, commonly used in the secondary treatment of the wastewater treatment plants, system in which the *Control y Supervisión de Procesos* research group in which I include myself has extensive experience toward getting an optimal plant control. We have also presented an optimization algorithm with promising results, based on *Swarm Intelligence* philosophy.

From a process control point of view, we have explored three topics of interest: agent-based modelling, swarm optimization and GPU (Graphics Processing Unit) computation.

Resumen

Con el tiempo y la paciencia se adquiere la ciencia.

Proverbio español.

El *modelado basado en agentes* (en inglés, agent-based modelling) es una técnica de modelado que ha revolucionado el campo del modelado y simulación de sistemas complejos, siendo una alternativa fiable a los tradicionales modelos de parámetros concentrados, fenomenológicos o empíricos. Esta técnica trata de obtener modelos de los sistemas a través de la representación de los elementos que componen dichos sistemas. Se trata de un modelado de abajo hacia arriba, donde las propiedades agregadas de los sistemas son consecuencia (emergen) de las propiedades de los elementos que los componen y de las relaciones entre estos elementos, los *agentes*. Esta técnica se ha aplicado satisfactoriamente en diferentes campos: economía, ciencias sociales, medicina, biología, etc.

El campo de la Ingeniería de Procesos y Sistemas se encarga de modelar procesos complejos cuya naturaleza dinámica hace que no suela ser una tarea sencilla, obteniéndose modelos no lineales, modelos en los que hay que tener en cuenta la presencia de perturbaciones, dependencias de estados pasados, etc. Esta técnica de modelado (modelado basado en agentes) se adecúa perfectamente al modelado de sistemas complejos. En esta tesis vamos, por lo tanto, a aplicar esta técnica al proceso de fangos activados, que tiene lugar en el proceso de depuración de aguas residuales. Los procesos de depuración de aguas residuales tienen una dinámica compleja debido a las variaciones en el caudal y composición del mismo a la entrada de las plantas de aguas residuales y la gran cantidad de procesos y microorganismos involucrados, lo cual hace que el control de estas plantas sea bastante complicado. Esto hace que el proceso de fangos activados sea un sistema idóneo para aplicar la técnica de modelado basado en agentes. El modelo desarrollado nos ha permitido comprobar la decisiva influencia que la composición y caudal de entrada a una planta de tratamiento de aguas residuales tiene en su dinámica. El modelo, una vez validado con respecto a una planta real, podría ser incorporado, por ejemplo, en un esquema de control predictivo basdado en modelos, donde las predicciones dadas por el modelo podrían ser usadas para obtener las acciones de control óptimas a aplicar al sistema (con respecto a ciertos objetivos de optimización). El problema de control de los sistemas de depuración de aguas tiene diferentes objetivos con distinta prioridad y que dependen además del diseño del sistema. Ejemplo de estos objetivos son: evitar fugas y caudales de aguas residuales fuera de los colectores, minimizar la energía de control (el coste energético de mover las válvulas y compuertas de regulación), maximizar el tratamiento aplicado para minimizar los contaminates presentes en el agua, reducir el volumen en los tanques en previsión de posibles tormentas futuras, etc [93]. Respecto al control del proceso de fangos activados, este consiste en mantener la concentración de sustrato bajo un cierto nivel establecido por ley, mantener la concentración de biomasa suficiente que permita la degradación del sustrato, y mantener el oxígeno suficiente que permita estas reacciones pero minimizándolo ya que supone un coste de energía. Las principales dificultades que presenta el control de este proceso son la falta de información medible en línea (se necesita emplear estimadores) y la propia complejidad del proceso (variaciones de carga y caudal, influencia meteorológica) [88].

En esta tesis se ha aplicado la técnica de modelado basado en agentes al proceso de fangos activados, que tiene lugar en una estación depuradora de aguas. Hemos estudiado la metodología y las diferentes opciones de implementación de este tipo de modelos, hemos desarrollado un modelo (una versión simplificada del proceso con un solo tipo de bacteria y un solo tipo de sustrato), y hemos desarrollado una notación que complementa al actual estándar de representación de este tipo de modelos.

Además, en esta tesis se ha trabajado en las técnicas englobadas bajo el nombre de *Swarm Intelligence*, que podríamos definir como inteligencia colectiva. Hace referencia a los patrones de comportamiento y comportamientos macroscópicos que son consecuencia de la colaboración de pequeñas entidades con reglas sencillas y sin una figura de liderazgo, por ejemplo, las colonias de hormigas, el vuelo de los pájaros, etc... Basado en esta filosofía, en cierto modo relacionada con los modelos basados en agentes, surgen un conjunto de técnicas *Swarm*, y en concreto en esta tesis hemos trabajado con algoritmos de optimización, probando el *Particle Swarm Optimization* y diseñando nuestro propio algoritmo, el *Hiker*, que presenta resultados muy prometedores, como se puede observar en este documento.

Habitualmente, en el campo de la ingeniería de sistemas y automática se persigue que los algoritmos y modelos se ejecuten lo más rápido posible para poder ser utilizados en tiempo real. Motivados por la necesidad de reducción de tiempos, en esta tesis hemos trabajado con técnicas de paralelización de código, y en concreto, código paralelo para ejecutar sobre tarjetas gráficas. Estas técnicas se han empleado tanto en el modelado basado en agentes como en los algoritmos de optimización *swarm*.

En resumen, se han aplicado las técnicas de modelado basado en agentes al proceso de fangos activados, uno de los procesos más utilizados en el tratamiento secundario en las plantas de depuración de aguas residuales, sistema en que el grupo de investigación de *Control y Supervisión de Procesos* en el que me incluyo tiene una amplia trayectoria y experiencia para obtener un control óptimo de la planta. Se ha presentado, asimismo, un algoritmo de optimización con resultados prometedores, basado en la filosofía *Swarm Intelligence*.

Desde un punto de vista de control de procesos, hemos explorado tres elementos de interés: modelado basado en agentes, optimización *Swarm* y computación en GPU (Graphics Processing Unit).

Objetivos

El objetivo general de esta tesis es aplicar la teoría de *Swarm Intelligence* y la técnica de modelado basado en agentes en el campo de ingeniería de sistemas y control automático, lo que implica explorar estas técnicas desde una nueva perspectiva, así como la aplicación de estas herramientas a sistemas donde no se han utilizado tradicionalmente. Los sistemas que se modelan en nuestro grupo de investigación son intrínsecamente dinámicos y complejos, por lo que la naturaleza de estos sistemas justifica la idoneidad de la aplicación de estas técnicas. Por lo tanto, los principales objetivos de esta tesis son:

1. Aprender y adquirir habilidades en la técnica de modelado basado en agentes. El primer objetivo comprende las etapas de investigación y documentación del estado del arte, la revisión de las propuestas de metodología de diferentes autores, así como ejemplos de aplicación en la literatura. La técnica se adquiere también con la práctica, lo cual se relaciona con el objetivo 4.

2. Seleccionar una plataforma software adecuada para desarrollar un modelo basado en agentes para el proceso de fangos activados.

3. Entender el proceso de fangos activados desde el punto de vista de sus constituyentes microscópicos y sus relaciones. Este objetivo tiene en cuenta la investigación en microbiología necesaria para entender el proceso antes de modelar.

4. Desarrollar un modelo basado en agentes para el proceso de fangos activados. Este modelo tiene por objeto representar la dinámica del proceso, desde el punto de vista de las entidades involucradas en el mismo. Se debe presentar una representación más realista del proceso que facilitará la comprensión del mismo. En este objetivo se utilizan los conocimientos adquiridos en los objetivos anteriores y se pone en práctica la técnica de modelado basado en agentes.

5. Utilizar el modelo basado en agentes desarrollado para tener una mejor comprensión del sistema y para generar nuevos conocimientos y conclusiones que podrían facilitar el control del sistema.

6. Desarrollar técnicas de optimización basadas en la filosofía *Swarm Intelligence* para mejorar la eficiencia en la búsqueda de soluciones a problemas de optimización. Este objetivo persigue además desarrollar un nuevo algoritmo de optimización, inspirado en el algoritmo *Particle Swarm Optimization.*

7. Probar las diferentes alternativas de programación en paralelo sobre tarjeta gráfica (toolboxes de MATLAB, código nativo CUDA) y seleccionar la más adecuada para las implementaciones de esta tesis.

8. Aplicar la computación en paralelo sobre tarjeta gráfica a los desarrollos de esta tesis para reducir el tiempo de ejecución de simulaciones y algoritmos. Este objetivo transversal incluye la aplicación de la computación en paralelo al modelado basado en agentes, la exploración de las diferentes posibilidades de uso, así como la paralelización del algoritmo *swarm* desarrollado.

El modelo basado en agentes y el algoritmo de optimización desarrollados son dos de las piezas claves de un esquema de control predictivo basado en modelos, en el cual el grupo de investigación tiene amplia experiencia. Para su aplicación en tiempo real, el tercer elemento serían técnicas de programación paralela sobre GPU para acelerar la ejecución del código. La integración de estos tres elementos (modelo, optimizador y ejecución sobre GPU) a un caso real queda ya fuera del alcance de esta tesis.

Tratamiento de aguas residuales

El capítulo 4 explica el proceso de depuración de aguas residuales, y en concreto el proceso de fangos activados (proceso que se ha modelado en esta tesis). El proceso de fangos activados es un tipo de proceso biológico que se lleva a cabo en el tratamiento secundario de las aguas residuales en una planta depuradora. En esta parte de la planta se trata de reducir la materia orgánica degradable biológicamente, de las aguas residuales industriales o urbanas. El proceso de lodos activos consiste en:

- Un tanque de aireación (reactor biológico) donde los lodos activos y la alimentación (contaminación o sustrato contenido en el agua) se mezclan y airean durante un cierto período de tiempo.

- Un sistema de separación del lodo activo y el efluente tratado (decantador secundario o clarificador).

- Una recirculación del lodo activo hacia el tanque de aireación.

- Un sistema de tratamiento y evacuación de los lodos producidos (purga).

Al reactor biológico o cuba de aireación llega el agua residual a tratar, donde tiene lugar el desarrollo de un cultivo biológico. La población bacteriana se debe mantener en un determinado nivel para llegar a un equilibrio entre la carga orgánica (sustrato) a eliminar y la cantidad de microorganismos (biomasa) en el reactor. Es necesario un sistema de aireación y agitación que proporcione el oxígeno necesario para la acción de las bacterias aerobias, que evite la sedimentación de los fangos en el reactor y que permita la homogeneización de los fangos activos. En este capítulo se introducen además ciertos conceptos de microbiología necesarios para el correcto entendimiento del documento. Además se incluye un estado del arte sobre modelado de tratamiento de aguas residuales, incluyendo modelado basado en agentes.

Metodología

Modelado basado en agentes

Los sistemas multiagente (MAS) son considerados tradicionalmente como un campo de la Inteligencia Artificial Distribuida (DAI), y junto con los sistemas basados en agente son una de las áreas de investigación más prometedoras dentro del área de Tecnologías de la Información y Computación. Actualmente no existe un consenso en la definición de agente. Las principales teorías al respecto se muestran en la sección 2.2.

Un **sistema multiagente** se puede definir como un conjunto de agentes autónomos, heterogéneos e independientes, que viven en un ambiente, interactuando con él y con los demás, con sus propios objetivos, capacidades y conocimientos.

Modelado basado en agentes es una metodología computacional que permite a los investigadores crear, analizar y experimentar con modelos compuestos de agentes que interactúan dentro de un entorno. La principal característica de estos sistemas es que existe una correspondencia directa entre las entidades en el sistema y en el modelo, y entre las interacciones en el sistema y en el modelo, como se muestra en la figura 2.2 en el capítulo 2. En este capítulo se incluyen además las ventajas y desventajas de esta metodología, las áreas de aplicación, un resumen sobre herramientas software para realizar este tipo de modelado, y una sección sobre estándares de representación.

Swarm Intelligence

Son una colección de algoritmos inspirados en la conducta social de los individuos en la naturaleza, dentro del campo de la computación evolutiva de la disciplina de inteligencia artificial (AI). Se trata de algoritmos basados en población inspirados por el comportamiento colectivo de sistemas descentralizados y auto-organizados, como las colonias de hormigas, termitas, abejas, y avispas, las bandadas de aves o bancos de peces. Este grupo de individuos con reglas simples que interactúan unos con otros, obtienen como consecuencia comportamientos colectivos emergentes o decisiones óptimas para el grupo, por ejemplo, dónde vivir, la formación de estructuras durante el vuelo, etc . En el capítulo 3 se presenta esta filosofía, ejemplos de aplicación, y se explica el algoritmo Particle Swarm Optimization, uno de los más famosos basados en este paradigma.

Computación paralela sobre tarjeta gráfica

En esta tesis se han aplicado técnicas de computación paralela sobre tarjeta gráfica para tratar de reducir los tiempos de ejecución de las implementaciones desarrolladas. Estas técnicas de paralelización se explican en el capítulo 5, mostrando un ejemplo sencillo, y los campos en los que se ha aplicado hasta la fecha.

Principales resultados del trabajo

En esta tesis se ha trabajado en acercar el campo de la ingeniería de sistemas y control automático al modelado basado en agentes y la teoría de Swarm Intelligence. Hemos estudiado el proceso de lodos activados desde el punto de vista de sus constituyentes microscópicos. Las conclusiones resultantes de este trabajo son:

- Hemos resumido los pasos de la metodología del modelado basado en agentes para que un recién llegado puede utilizar el capítulo 2 como guía inicial sobre el tema.

- Hemos propuesto una nueva notación SSABR para representar modelos basados en agentes, que está más cerca del campo de la ingeniería de sistemas y atestigua el carácter dinámico del comportamiento de los agentes a través de una representación en el espacio de estados. Se han incluido ejemplos de aplicación para ilustrar la utilidad y aplicación del SSABR. Esta notación se podría utilizar para enseñar modelado basado en agentes en ingeniería y también para compartir resultados de investigación, junto con el protocolo ODD.

- Hemos desarrollado un modelo basado en agentes para el proceso de fangos activados, que permite probar la influencia de diferentes protocolos de operación del reactor y una población de bacterias configurable.

- Hemos confirmado la influencia significativa que las oscilaciones y las variaciones en la composición del caudal de entrada a una estación depuradora de aguas residuales tienen en un proceso de fangos activados. Nuestro modelo permite probar diferentes escenarios para estudiar la dinámica del sistema.

- Hemos proporcionado la posibilidad de lanzar cientos de simulaciones en paralelo, para estudiar, por ejemplo, la influencia de la estocasticidad de la población bacteriana en el comportamiento macroscópico del sistema.

- Hemos desarrollado un algoritmo de optimización, el *Hiker*, inspirado en la teoría de *Swarm Intelligence*, con resultados prometedores de en rendimiento.

Contents

List of Figures

List of Tables

Part I

Introduction

This part contains the introduction, background, objectives, and document structure.

Chapter 1

Introduction and Objectives

The more extensive a man's knowledge of what has been done, the greater will be his power of knowing what to do.

Benjamin Disraeli

Summary: This chapter introduces the motivation and establishes the objectives of this thesis.

1.1 Motivation and Background

This work has been carried out within the Process Control and Supervision (CSP) research group of the University of Valladolid. Members of this group mainly belong to the System Engineering and Automatic Control department.

The activities of the CSP research group are classified into several research lines, where the *process modelling and simulation* research line is the relevant one for the scope of this thesis. This is one of the main research areas of the group, with wide experience in sugar processes, petrochemical processes, wastewater treatment processes and renewable energies. Main topics are:

- Real-time process simulators.
- Distributed simulation.
- Development of process libraries.
- Languages for automatic modelling.

Thus, the research group makes its best efforts in developing models with a high level of detail that will be used afterwards in simulators or internally for model based controllers or for model supervision (e.g. fault diagnosis detection).

In certain type of processes, such as chemical reactors or biological processes, where the macroscopic models are usually obtained from empirical knowledge, a bottom-up approximation would obtain better results in terms of the representation of the underlying process.

Some members of the group have been working, in cooperation with a research group of the University of Salamanca, in the project *Técnicas avanzadas de supervisión y control para la operación óptima de EDARS* (ref. $DPI2006 - 151716 - C02 - 02$), where the objective was to apply advanced techniques for control and supervision of wastewater treatment plants.

One of the main biological processes in a wastewater treatment plant is the activated sludge process. This is an adequate test bench where modelling from a microscopic level would bring a better understanding of the process dynamics (and consequently a better control). Large variations in the influent wastewater flow rate, concentration and composition make wastewater treatment processes inherently dynamic. These variations are difficult to control and the adaptive behaviour of the involved microorganisms imposes further difficulties in terms of time-varying process parameters. Computer models and simulations are necessary to describe and control these systems, but this can result in highly complex models, sometimes difficult to apply from an operational point of view.

Modelling from the microscopic entities point of view is encompassed in the Agent-Based Modelling or Individual-Based Modelling techniques. This is the main topic of this dissertation, from the point of view of the system engineering field. Agent-based modelling is a suitable tool to study complex systems, formed by many interacting entities and non-linear interactions among them. They may have several stationary states, the resulting outcome may depend on the previous history and perturbations, they behave in non-stationary ways, show periodic or non-periodic oscillations, etc.

A collection of microscopic individuals with simple behaviours can be seen as a *swarm*. This swarm can exhibit some kind of intelligence at a macroscopic level (e.g. flock of birds). Taken into account the experience of the research group in the artificial intelligence fields, it is also interesting to explore what is called *Swarm Intelligence*, as it is related, somehow, with Agent-Based Modelling. Particle swarm optimization is a powerful tool, easy to understand and to implement, and also computationally efficient. This qualities encourages us to use this tool and have inspired us to develop our genuine idea of the *Hiker* algorithm.

1.2 Objectives

The general objective of this dissertation is to apply the Swarm Intelligence and Agent-Based Modelling (ABM) techniques to the system engineering and automatic control field, which implies exploring these techniques with a new perspective and also applying these tools where they have not been traditionally used. The systems that are modelled in our research group are intrinsically dynamic and complex, so the nature of these systems justifies the suitability of application of these techniques.

Thus, the main objectives of this thesis are:

1. To learn and acquire abilities in the agent-based modelling technique. The first objective encompasses the steps of research and documentation of the state of the art, reviewing the methodology proposals from different authors as well as application examples in the literature. The technique is acquired with practise, which is related with objective 4.

2. To select a suitable software platform to develop an agent-based model for the activated sludge process.

3. To understand the activated sludge process from the point of view of its microscopic constituents and its relationships. This objective takes into account the research to be done in microbiology to understand the process before modelling it.

4. To develop an agent-based model for the activated sludge process. This model aims to represent the dynamics of the process from the point of view of the entities involved in the process. It should present a more realistic representation of the process that also facilitates the understanding of the system. This objective groups the knowledge acquired from previous objectives and put in practise the agent-based modelling technique to develop a model of the activated sludge process.

5. To use the developed ABM to have a better understanding of the system and to generate new insights and conclusions that could facilitate the control of the system.

6. To develop optimization techniques based on the philosophy of swarm intelligence to improve efficiency in the search for solutions to optimization problems. This objective aims to apply the swarm intelligence to develop a new optimization algorithm, inspired in the Particle Swarm Optimization.

7. To test the different alternatives of parallel programming over GPU (Graphic Processing Units) (MATLAB toolboxes, CUDA native code) and select the most adequate for the thesis implementations.

8. To apply the parallel computing over GPU technique to the developments of this dissertation to reduce the execution time of simulations and algorithms. This transversal objective includes the application of parallel computing to agent-based modelling, exploring the different possibilities of use, and also parallelizing the swarm intelligence algorithm developed.

The agent-based model and the optimization algorithm developed as part of this work are two of the main blocks of a model based predictive controller, where our research group has wide experience. For its application in real time, the third element would be the parallel programming techniques over GPU for speeding up code execution. Integration of these three elements (model, optimizer and execution over GPU) and application to a real case is out of the scope of this thesis.

1.3 Document Structure

The document is organized in four parts: part one introduces the work developed in this dissertation and states its objectives, the second one summarizes some fundamental concepts to facilitate the comprehension of the thesis; the third one exposes the developments of this dissertation, and the forth includes the conclusions and future work directions.

Inside part two, chapter 2 introduces Agent-Based Modelling, the principal progress about this methodology, the application areas, the strengths and weaknesses, the software used in this thesis, and the main representation standard. Chapter 3 explains some fundamentals about Swarm Intelligence, exposes applications of this paradigm, and introduces the Particle Swarm Optimization algorithm. Chapter 4 introduces the field of wastewater treatment, the principal models used in the field, the activated sludge process (process used in this dissertation), and also explains some microbiological concepts needed when describing the activated sludge process from the point of view of its microscopic components, bacteria. Chapter 5 deals with fundamentals about parallel computing using graphic processing units (GPUs).

In part three, chapter 6 presents our proposal of representation for agent-based models; an example of application is introduced in the chapter, and three additional examples in appendix A. The next chapter 7 covers the agent-based models for the activated sludge process, outcomes of this dissertation. It also includes our efforts in parallelizing the model to improve its performance. The next chapter 8 presents a new optimization algorithm based on the swarm intelligence theory.

In part four, chapter 9 exposes the main conclusions of this dissertation (section 9.1), collects the contributions during this thesis (section 9.2), and presents some future work directions (section 9.3).

Part II

Fundamentals

In this part of the thesis, some theoretical notions about the techniques and the application field of this thesis are introduced. This part is divided in four chapters with fundamentals about agent-based modelling, swarm intelligence, wastewater treatment, and parallel computing on GPU.

Chapter 2

Agent-Based Modelling

Learn from yesterday, live for today,
hope for tomorrow. The important thing
is to not stop questioning.

Albert Einstein, *Relativity: The Special and the General Theory.*

Summary: This chapter is focused on agent-based modelling, the central methodology of the thesis. We briefly introduce this concept and give some historical background. We then explain the structure of the methodology and the key aspects of application including a short introduction to the programming environments used in this thesis.

2.1 Introduction

Agent-based modelling (ABM) is a modelling approach that has gained increasing attention over the past years. This methodology has become so widespread due to the increasingly complex world where we live. We have the chance to analyse more and more complex systems than before, where traditional models are no longer applicable, for example, the deregulation of the electric power industry, as Macal and North [76] show. Doyne Farmer and Duncan Foley encourage a revision of traditional models in their article *The economy needs agent-based modelling* [29], where they summarize this need frankly and with no reservations in the sentence "the leaders of the world are flying the economy by the seat of their pants".

In the past, the models where limited by mathematical tractability, models needed to be simple enough to be solved mathematically. This limitation has been solved due to computer simulation. Nowadays we live in the so called information society, where more and more data are available, organized into databases at finer levels of granularity. And also the computational power of computers continues increasing, providing the scientific community with more and more support to compute large-scale simulation that were not plausible years before.

ABM has historical roots in the developments of automata approaches and complex adaptive systems (CAS) and the underlying notion that "systems are built from the bottom-up" [84]. The CAS field studies the arise of complex behaviours, adaptation and emergence of complex systems.

Before we start talking about ABM, it is essential to clarify what an agent and agent-based modelling are. Thus, next section 2.2 is about definitions. Then, in section 2.3 we review the application areas of this methodology. Section 2.4 summarizes the principal strengths and weaknesses of ABM. Section 2.5 exposes some guidelines about how to develop agent-based models. Section 2.6 exposes a brief review of ABM tools. The last section 2.7 reviews the current standards in the field.

2.2 Definitions

Multi-agent systems (MAS) are traditionally considered a sub-field of Distributed Artificial Intelligence (DAI) field, and together with agent-based systems are one of the most promising investigation areas inside the information technologies and computation field.

Although the term **agent** is widely used, there is no universal accepted definition inside the multi-agent field in general, neither the agent-based modelling community in particular. The term *agent* appears in a diverse range of sub-disciplines of information technology with slightly different meanings. There are many articles that try to clarify what characteristics an agent should own to be considered a so called *agent*. The same occurs with the term "multi-agent system". In order to take a further look into this subject we do recommend reading [32, 46]. In the following, we summarize some definitions.

From a practical standpoint, an **agent** can be defined as Macal and North [75] show:

- An agent is identifiable, a discrete individual with a set of characteristics and rules governing its behaviours and decision-making capability. Agents are self-contained. The discreteness requirement implies that an agent has a boundary and one can easily determine whether something is part of an agent, is not part of an agent, or is a shared characteristic.

- An agent is situated, living in an environment with which it interacts along with other agents. Agents have protocols for interaction with other agents, such as for communication, and the capability to respond to the environment. Agents have the ability to recognize and distinguish the traits of other agents.

- An agent may be goal-directed, having goals to achieve (not necessarily objectives to maximize) with respect to its behaviours. This allows an agent to compare the outcome of its behaviour relative to its goals.

- An agent is autonomous and self-directed. An agent can function independently in its environment and in its dealings with other agents, at least over a limited range of situations that are of interest.

- An agent is flexible, having the ability to learn and adapt its behaviours based on experience. This requires some form of memory. An agent may have rules that modify its rules of behaviour.

There are several features that are common to most agent theories. These are: autonomy, heterogeneity, active/pro-active, reactive, bounded rationality, communicative, learning... An explanation of these features can be found at [54].

According to Torsun [126], a **multi-agent system** consist of autonomous, heterogeneous and independent agents, living in an environment, interacting with it and with each other with their own goals, capabilities and knowledge.

Multi-agent systems are applied in many disciplines. Luck et al. [72] classify the main agent research areas in three categories: agents as design metaphor (as a way of structuring software development), agents as a source of technologies (including different techniques and algorithms for dealing with interactions in dynamic, open environments), and agents as simulation (for representing complex dynamic systems. Agent-based modelling is included in this category), as Figure 2.1 summarizes.

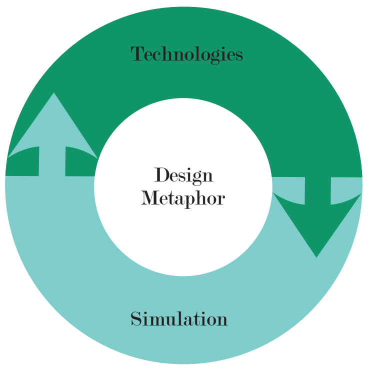


Figure 2.1: Agent research areas. Based on [72].

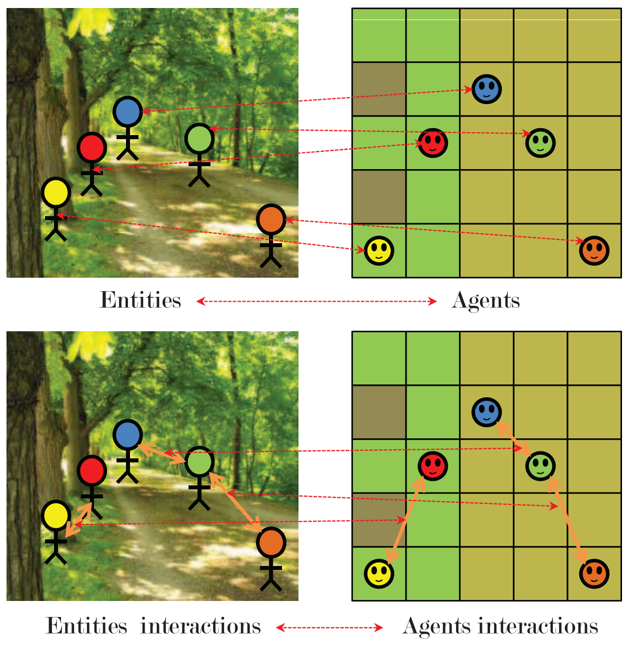


Figure 2.2: Correspondence between the modelled system (left) and ABM (right). Based on [33].

Agent-based models consist of agents that interact within an environment. Formally, ***agent-based modelling*** *is a computational method that enables a researcher to create, analyse and experiment with models composed of agents that interact within an environment* [34]. The models represent the simultaneous operation and interactions between multiple agents in an attempt to recreate the behaviour of a complex phenomenon.

Models are often aimed at obtaining insights about some aspects about a given real system for a specific purpose. In order to facilitate this understanding and focus on the gist of the process, parts of the complexity of the target system are intentionally lost in the construction of a model. For instance, when modelling the wind resistance of a car, we deliberately do not consider the colour or the scent of the piece, we just pay attention to those aspects of the real system that are considered essential and we disregard the rest of the details. This simplifying process, retaining only those features that are considered relevant and making assumptions about unknown aspects, is called **abstraction**. However, as Galán et al point out [33], if an abstraction does not in any way represent its modelling target, it would be inappropriate to call it a model. In contrast with other modelling paradigms in which it is not always easy to find the link between the analysed system and its modelled representation, ABM facilitates this process since there is a direct correspondence between entities in the model and entities in the modelled system, and interactions in the model and interactions in the modelled system, as Figure 2.2 shows.

A review of the literature reveals many different terms being used to describe what we have in this thesis so far called *agent-based modelling*. These terms include agent-based modelling [10], multi-agent-based simulation [22], individual-based modelling [54, 52, 41], etc. An interesting attempt to clarify these terms is conducted by Hare and Deadman [46]. Also Heppenstall et al. show at [54] the differences between Cellular Automata, individual-based models and microsimulation (MSM): *A CA is a discrete dynamic system, the behaviour of which is specified in terms of local relations. The space in a CA system is divided into a lattice or grid of regularly-space cells of the same size and shape, usually square. Each cell has a value either 0 or 1 or on a scale from 0 to 1. The state of a cell and its behaviour is determined by the state of other cells in close proximity at a previous time step, by a set of local rules and by the cell itself. An important feature of a CA is that the automata's location does not move; they can only change their state. The position of the cells and their neighbourhood relations remain fixed over time. In contrast, agents can be either fixed in location or free to 'roam' around their environment. Unlike agents, CAs cannot have more than one attribute; for example, a cell could be occupied or unoccupied, but the cell could not contain multiple attributes such as building type, date built, etc. MSM is a well established methodology that works on the principle of creating small area microdata at a point in time, and then generating future microdata from that basis. Crucially in contrast to ABM, MSM only models one-direction interactions: the impact of the policy on the individuals, but not the impact of individuals on the policy and interactions between individuals are not simulated. Furthermore MSM models do not have the behavioural modelling capability of ABM.* For these authors, individual-based modelling (IBM) is a category grouping ABM, CA and MSM; it is not so in Ecology, where IBM is used as equivalent to ABM.

Agent-based modelling can be presented as "magic", but it is not more than a formal model, that, in fact, can be expressed as a compact set of mathematical equations (although this work is not trivial). Leombruni and Richiardi [68], present a mathematical formalism able to describe both traditional analytical models and agent-based ones. Once a model has been completely defined, we can derive the logical implications consequence of the assumptions of the model. This process is called inference. Many formal models typically infer these results deductively, however, in the case of ABM, models are often so complicated that it is not feasible to obtain compact close form solutions to explore the implications. In these cases we have to resort to computer simulation as inference tool. From the subset of the simulated cases modellers inductively derive general properties of the results. Since in the majority of the models the study is done by simulation, the term agent-based modelling and agent-based simulation are sometimes used interchangeably. As Galán and Izquierdo [33] point out, *it is important to realise that a computer program is a formal model that can be expressed in mathematical language, e.g. as a set of stochastic or deterministic equations, and computer simulation is an inference tool that enables us to study it in ways that go beyond mathematical tractability.*

A number of researchers think that the alternative to ABM is traditional differential equation modelling, but they should be seen as complementary approaches to the study of formal systems. Agent-based modelling and simulation allows to explore the properties of certain formal models that are intractable using traditional formal analysis, providing new knowledge, giving us the opportunity to explore inductively models which are difficult to solve analytically.

An important characteristic of complex systems that can be captured using ABM is the **emergent phenomena**, that is, stable macroscopic patterns arising from local interaction of individual entities. By definition, emergent phenomena cannot be reduced to the systems' parts; the whole is more than the sum of the parts. It is said that the process to represent (macroscopic level) emerges from the microscopic level, i.e., the interaction of relatively simple behaviours leads to complex behaviour from a global perspective. Phenomena such as flocks of birds, school of fish, and the complex biological systems of cells are good examples of how systems with simple goals can show complex emergent behaviours as a result of the communication with neighbouring agents.

One of the first models illustrating the emergent phenomena is the Boids Model, developed by Craig Reynolds [109], which simulates the flocking behaviour of birds. In this model, each agent is governed by three rules: separation, alignment and cohesion, that lead to a coordinated flight.

2.3 Application areas

Bonabeau [13] has identified a list of conditions in which it is appropriate to use ABM to capture the emergent behaviour:

- Agents exhibit complex behaviour, including learning and adaptation in their behaviour.
- The behaviour of individuals cannot clearly be defined through aggregate transition rates.
- Agents interactions are complex, non-linear and present heterogeneity, and the interactions are dynamic.
- The modelled population is heterogeneous.

- Agent behaviour is stochastic. Points of randomness can be applied strategically within agent-based models, rather than a noise term added more or less arbitrarily to an aggregate equation.

- When it is important that agents form organizations or groups, and when it is relevant to model adaptation and learning not at individual level but at organization level.

- Agents spatial distribution is important.

ABMs have been developed for a wide diverse range of subject areas, from economics, ecology, social science, microbiology, environmental modelling, etc. Reviews of different applications can be found in almost any introductory guide to ABM. To see some examples of application areas the reader can review: [54, 75, 46, 125].

In [75] we can find the following summary (table 2.1) of ABM applications.

Business and Organizations	**Society and Culture**
Manufacturing Operations	Ancient civilizations
Supply chains	Civil disobedience
Consumer markets	Social determinants of terrorism
Insurance industry	Organizational networks
Economics	**Military**
Artificial financial markets	Command and control
Trade networks	Force-on-force
Infrastructure	**Biology**
Electric power markets	Population dynamics
Transportation	Ecological networks
Hydrogen infrastructure	Animal group behaviour
Crowds	Cell behaviour and subcellular processes
Pedestrian movement	
Evacuation modelling	

Table 2.1: Agent-based Modeling Applications. From [75]

As Tesfatsion [125] points out in his online repository about ACE (Agent-based Computational Economics), there are four strands differentiated by objective in which ABM research can be divided. These are: empirical understanding, normative understanding, qualitative insight and theory generation, and methodological advancement. This implies that ABM can be used to explain systems, extend the understanding of the system, and also to generate new knowledge about systems.

2.4 Strengths and weaknesses

The mayor advantages of ABM are [11, 13, 27]:

- It facilitates the abstraction since there is a direct correspondence between entities in the model and entities in the modelled system, and interactions in the model and interactions in the modelled system.
- It allows the representation of interactions and spatial distribution of entities and also the study of the bidirectional relations between individuals and groups.
- It provides the possibility to model heterogeneity almost by definition.
- It leads to formal yet more natural and transparent descriptions of the target system.
- It improves the understanding of systems.
- It can capture emergent behaviour.
- It can be used as virtual experiments in order to predict the behaviour of a system under certain conditions, that is, it is suited for detailed hypothesis-testing, i.e. for the study of the consequences of hypotheses regarding the interactions of agents.

There are also some weaknesses in applying this modelling approach:

- ABMs are very sensitive to initial conditions and to small variations in interaction rules, this path dependence means that using ABM for prediction can be challenging [49, 54].
- In most of the cases, the models cannot be solved deductively, as a consequence simulations must be run to study the behaviour of the model [13].
- The process of calibrating an ABM with the correct parameter values to fit the behaviour of the target system is challenging; it is usually necessary to use optimization tools.
- These models can have a high computational load.

2.5 Methodology

The process of developing an ABM is carried out by three different roles according to Drogoul et al. [22, 25, 33]: the *thematician*, the *modeller*, and the *computer scientist*.

The *thematician*'s work is to abstract the target system identifying the purpose of the model and hence the different individuals of the target system and its relationships. In order to fulfil the task, the thematician has to be an expert in the domain but not necessarily a modelling expert. Consequently, the conceptualization of the model (abstraction step), output of this stage, is usually a non-formal model in which many of the processes are not fully specified. The following is the design step, where the *modeller* transforms this conceptualization into a complete and formal set of specifications. Finally, the approximation process is conducted by the *computer scientist*, which must find a suitable (formal) approximation to the modeller's model that can be executed in a computer. The goodness of the approximation may be quantified by the effect in the results, the less impact the better.

The modelling scheme described upwards can be embedded in a more general methodology which includes the design and development of the ABM model, but also its proper use, e.g. the simulation, analysis, validation or interpretation of the model. Although the details can depend on the context and goals of the model, typically this approach involves five different steps. Firstly, the definition of the agents involved, their characteristics, the environment they live in (subsection 2.5.1), and the relationships between all of them (subsection 2.5.2). Even though in practice, research teams do not divide clearly their work according to the diverse roles and very often several roles may correspond to the same person, these tasks may be attributed to the thematician and the modeller. The following is the model implementation by the role of computer scientist using a software platform (subsection 2.5.3). The next is the verification, calibration and validation processes (subsection 2.5.4). Finally, simulations can be performed (subsection 2.5.5).

2.5.1 Agents and their Environment

The first step is to identify the agent types along with their attributes. The agent types are the different species of individuals (entities in general) playing an important role in the model. The agents are characterized by their attributes. Attributes can be grouped into:

- Spatial location: identifying the position of the agents in the environment. Sometimes it is not used; for example, if the topology of the agents is a network, or if the spatial position of the agents is not relevant.
- Appearance characteristics: for visualization purposes. Agents can have different visualization properties, such as: shape, colour, visibility, label, trail, etc.
- Custom attributes, depending on the system to be modelled, such as: age, temperature, sex, etc.

The agents can have two types of attributes, static ones (also called *parameters*), and dynamic ones that change over time (also known as *agent states*).

Some ABM [124] add learning processes to the agents, making use of different techniques. These include reinforcement learning algorithms, neural networks, genetic algorithms, genetic programming, and a variety of other algorithms that attempt to capture aspects of inductive learning.

Macal et al. [76] group the relationships between agents in five categories according to its spatial distribution:

1. Soup. A nonspatial model in which agents have no locational attribute.
2. Grid or lattice. Cellular automata represent agent interaction patterns and available local information by a grid or lattice; cells immediately surrounding an agent are its neighbourhood. Geographic information systems (GIS) of raster type, where agents move over realistic geospatial landscapes [34], are included in this category.
3. Euclidean space. Agents roam in 2D or 3D spaces; vectorial GIS can also be used [34].
4. Networks. Networks may be static (links pre-specified) or dynamic (links determined endogenously). Sometimes a multidimensional model can be developed, where different network layers can affect each other; an example of these networks is multiplex networks [45].

They also emphasize the idea that the fact that agents only interact with a limited group of agents in their surroundings is most important than the spatial distribution of agents itself.

2.5.2 Agent Methods and Interactions

In this subsection the most common process for the specification of the methods of updating the agents attributes, in response to the interaction with other agents or the environment is exposed, although there can be other modelling paradigms, such as SDML (strictly declarative modelling language) [89].

Agents can interact with each other and amongst themselves and with the environment. Relationships may be specified in a variety of ways, from simply reactive (i.e. agents only perform actions when triggered to do so by some external stimulus e.g. actions of another agent) to goal-directed (i.e. seeking a particular goal) [54]. The methods can be classified into:

- Movement methods: jump, turn...
- Appearance change: size, colour, shape change...

- Custom changes according to the agents attributes.

These methods are based on published literature, expert knowledge or data analysis [54]. They are also known as rules or submodels in the literature depending on the field.

Moreover, the methods to control which agents interact, the order and timing must be specified. These methods usually follow if-else statements based on the agents states (values of their attributes) carrying out an action once a specified condition has been satisfied.

To develop these methods, parameters related to the agents and the environment must be defined.

Both Macal and North [76] and Railsback and Grimm [106] present a set of questions to be answered before an ABM is designed. Table 2.2 shows the version of Macal and North.

2.5.3 Implementation

Once you have the executable model [33], that is, having completed the modelling steps of abstraction, design and approximation, the next step is the implementation of the model in a computational software, using a programming language or using a higher-level agent-based toolkit. There are several software platforms available which will be mentioned in section 2.6.

The model implementation is composed by several parts:

- Memory allocation, initialization of parameter values and agents states.
- The program would iterate between the different actions and through all the agents, sequentially checking if the requirements to happen are met; if then, the action occurs. For example, if an agent reaches a fixed value for reproduction, it could reproduce as a function of a statistical distribution if the model is stochastic.
- Calculate the *aggregated variables* that represent the macroscopic emergent behaviour of the model. These variables are expressed by means of relationships between the agent states and the parameters of the agents and of the environment.
- At every iteration, time must be evolved. If time is modelled by discrete time steps, an step counter is incremented.
- It is checked if the termination condition is satisfied.

The algorithmic description is shown in figure 2.3.

The behaviour of the agents can be scheduled to take place **synchronously** (i.e. every agent performs actions at each discrete time step, all changes occur simultaneously), or **asynchronously** (i.e. agent actions are scheduled

Model Purpose and Value-added of Agent-based Modelling:
What specific problem is the model being developed to address?
What specific questions should the model answer?
What kind of information should the model provide to help make or support a decision?
Why might agent-based modelling be a desirable approach?
What value-added does agent-based modelling bring to the problem that other modelling approaches cannot bring?
All About Agents:
What should the agents be in the model?
Who are the decision makers in the system?
What are the entities that have behaviours?
Where might the data come from, especially on agent behaviours, for such a model?
Agent Data:
What data on agents is simply descriptive (static attributes, parameters)?
What agent attributes are calculated endogenously by the model and updated for the agents (dynamic attributes, states)?
What is the agents' environment?
How do the agents interact with the environment?
Is agent mobility through space an important consideration?
Agent Behaviours:
What agent behaviours are of interest?
What decisions do the agents make and what information is required to make such decisions?
What behaviours are being acted upon?
What actions are being taken by the agents?
How would we represent the agent behaviours? By If-Then rules? By adaptive probabilities, such as in reinforcement learning? By regression models or neural networks?
Agent Interactions:
How do the agents interact with each other?
How do the agents interact with the environment?
How expansive or focused are agent interactions?
Agent Recap:
How do we design a set of experiments to explore the importance of uncertain behaviours, data and parameters?
How might we validate the model, especially the agent behaviours and the agent interaction mechanisms?

Table 2.2: Questions to Ask Before Developing an Agent-based Model [106]

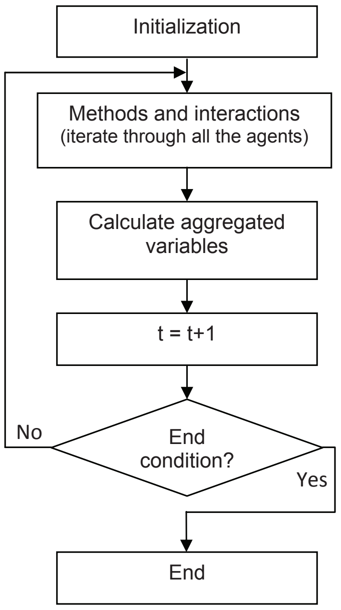


Figure 2.3: Flowchart of a generic agent based model scheduling.

by the actions of other agents, and/or with reference to a clock) [54]. In the model's schedule is also important to define how variables are updated, asynchronous (a state variable is immediately assigned a new value as soon as that value is calculated by a process) or synchronous (the new value is stored until all agents have executed the process, and then all are updated at once). Most ABMs represent time simply by using time steps, but time can be represented in other ways [43].

2.5.4 Verification, validation and calibration

Once there is a first working version of the model, it must be checked whether it accomplishes the objectives for what it was designed.

The first set of tests is called verification, also called internal validation [123, 9, 22], program validation [110] or face validation [64]. It is checked that the simulation performs as planned to do. Here some coding errors (called 'errors') can appear or design errors or modelling mismatch (called 'artefacts') [33] may appear. When this occurs, the results of the simulations may be erroneous. A great recommended read on the subject is [33].

Then, the process of validation must be performed, ensuring that the behaviour of the model does correspond to the behaviour of the system modelled.

If there is the need to adjust the model to a real case, there must be a process of calibration where the values of the parameters are tuned so that the output of the model fits real data.

These three processes are explained in more detail in [64].

2.5.5 Simulation

A simulation is the process of running an experiment in a computational model. A set of entries is given, which are the initialization values of the states and the values of the parameters. These values depend on the experiment that is going to be performed. After the experiment, the data generated as a result must be analysed.

If the model is stochastic, two agent-based simulations can generally bring different results even if the underlying model is exactly the same.

2.6 Software

Nowadays, there are many software options for implementing an ABM. The general criteria for selecting one or another are: the easiness of developing the model/using the system; the size of the community using the system; the availability of help or support (most probably from the user community); the size of the community familiar with the programming language in which the system is implemented (if a programming language is necessary to implement the model); if the system is still maintained and/or updated; the availability of demonstration or template models; the technical and how-to documentation; and the licensing policy (open source, shareware/freeware, or proprietary). There are other criteria related to the ABM modelling process: number of agents that can be modelled; degree of interaction between agents; ability to represent multiple organisational/hierarchical levels of agents; variety of model environments available (network, raster, and vector); possible topological relationship between agents; management of spatial relationships between agents, and agents with their environment; mechanisms for scheduling and sequencing events, etc. All these topics are covered in the book *Agent-Based Models of Geographical Systems* [54] under the chapter Guidelines for Choosing a Simulation/Modelling System, and in the article (in Spanish) *Modelado basado en agentes para el estudio de sistemas complejos* [98], and we also recommend to review [50, 54, 75, 76] to have a more extended knowledge on the subject.

Open source software is that whose source code is public and, enabling anyone to copy, modify and redistribute the system without paying royalties. Under this classification we can find well known toolkits: MASON [73], Repast [18], Swarm [86] (Swarm was the first ABM software development environment, launched in 1994 at the Santa Fe Institute.), SeSAm [63], NetLogo [129], etc. Proprietary software is developed by an organisation that exercises control over its distribution and use and requires a licence

(AgentSheets [108], AnyLogic [7]).

An ABM can also be implemented using spreadsheets, general computational mathematical systems: (MATLAB [81], Mathematica [131]) or general programming languages.

In this thesis we have tested three ABM environments: SeSAm, AndroMeta [6] and NetLogo.

SeSAm is short for Shell for Simulated Agent Systems. It is a generic environment for the development and simulation of Multi-Agent models. The main focus is to enable scientists to construct models by visual programming. The main advantage of multi-agent simulation is that the agent paradigm is very intuitive, especially when modelling societies. Moreover it has some valuable properties, like the possibility for formulating flexible interaction between agents, multi-level interaction, adaptivity, etc. SeSAm was developed at the University of Wurzburg and applied in several projects in different application domains. SeSAm is open source (LGPL) and available to download for free.

AndroMeta is a software platform for technical and scientific computing which spans a diverse range of fields: from machine learning and artificial intelligence (AI) in general, distributed and concurrent computing, language design, modelling and simulation, etc, all of them are unified into an advanced yet easy-to- use C++ framework. The website of the manufacturer is not reachable nowadays.

NetLogo is an open source programmable modelling environment for simulating natural and social phenomena. It was authored by Uri Wilensky in 1999 and has been in continuous development ever since at the Center for Connected Learning and Computer-Based Modeling. It is used by tens of thousands of students, teachers and researchers worldwide. It also powers HubNet participatory simulations. It comes with an extensive models library including models in a variety of domains, such as economics, biology, physics, chemistry, psychology, system dynamics... NetLogo allows exploration by modifying switches, sliders, choosers, inputs, and other interface elements. Beyond exploration, NetLogo allows authoring of new models and modification of existing ones.

2.7 Standards

The most popular standard for describing an agent-based model is the ODD Protocol, proposed by Volker Grimm et al. [42]. Its objective is to standardize the descriptions of individual-based and agent-based models, to make model descriptions more understandable and complete, thereby making ABMs less subject to criticism for being irreproducible. The ODD is organized around the three main components to be documented about a model: *Overview, Design concepts, and Details*; these sections must be written in a certain order. These sections encompass seven elements that must be documented in sufficient depth for the model's purpose and design to be clear and replicable for a third party: Purpose, State Variables and Scales, Process Overview and Scheduling, Design Concepts, Initialization, Input, and Submodels.

- The *Overview* section aim is to sketch the model, describing: the entities in the model, the methods and interactions, and the model's schedule. The Purpose subsection is intended to explain the goal of the model and its future use and application. The State variables and scales subsection outlines the structure of the model at a high level, but also at a low level, specifying all the variables that constitute the state of the model. The Process overview and scheduling subsection lists all the processes that occur in the model and how they are scheduled.

- The *Design* concepts section does not describe the model itself, but the general concepts about it that are characteristic of ABMs. They group some of the following subsections (not necessary all): Basic principles, Emergence, Adaptation, Objectives, Learning, Prediction, Sensing, Interaction, Stochasticity, Collectives, and Observation.

- The *Details* section should allow a complete re-implementation of the model. Initialization describes the starting state of the model. Input data describes any outside data that is input into the model. The Submodels subsection explains in detail the processes listed in the Process overview and scheduling subsection within the *Overview* section.

In addition to the original 2006 publication, Grimm et al. have continued to publish updates to the protocol with examples of its application to research projects [43].

In the literature, another proposal to describe an agent based model can be found; Hinkelmann et al. [56] propose an addition to the ODD protocol: an algebraic structure to describe an ABM as a dynamical system using polynomials.

In this thesis we propose a complementary approach for representing agent based models. Our notation provides a new approach for representing the agents in the model, the environment, and the relationships between agents with other agents, and agents with the environment, which lead to different patterns and emergent behaviours. We propose a notation inspired by the State Space representation of systems, since this work has been developed in the System engineering and automatic control field and from our point of view, agents are dynamical systems whose states evolve in time within a state space domain, as a result of complex relations between agents and agents with the environment. *The State Space Agent-Based Representation* will be described at chapter 6.

In the next chapter...

In the next chapter we are going to introduce the concept of Swarm Intelligence in the context of this thesis.

Chapter 3

Swarm Intelligence

It is possible to make things of great complexity out of things that are very simple. There is no conservation of simplicity.

Stephen Wolfram

Summary: In this chapter the fundamentals of Swarm Intelligence are reviewed. Also the Particle Swarm Optimization algorithm and the variation used in this thesis are explained.

3.1 Introduction

Swarm Intelligence is a collection of algorithms inspired by the social behaviour of individuals in nature, within the field of evolutionary computation inside the artificial intelligence (AI) discipline. They are population based algorithms inspired by the collective behaviour of decentralized and self-organizing systems. Examples of such systems which have fascinated scientists for many years are ants, termites, bees, and wasps, the flocks of birds or schools of fish. This group of individuals with simple rules that interact with each other produce, as a consequence, emergent collective behaviours or decisions that are optimal for the group, for example, where to forage, where to live, the formation of structures during the flight, etc... A good introduction to Swarm Intelligence with explanation of several natural collective behaviours can be found at [12].

As in the agent-based modelling approach, the foundation of this paradigm is the collective behaviour of decentralized individuals with predefined rules that cooperate. Even though the single members of these colonies (swarms) are non-sophisticated individuals, they are able to achieve complex tasks in cooperation. Coordinated colony behaviour emerges from relatively simple actions or interactions between the colonies' individual members.

A **swarm** can be defined as a structured collection of interacting organisms (or agents). **Swarm Intelligence** can be defined as 'the emergent collective intelligence of groups of simple agents' [14]. '**Self-organization** is a set of dynamical mechanisms whereby structures appear at the global level of a system from interactions of its lower-level components' [14].

Mark Millonas [85], at Santa Fe Institute, who develops this kind of swarm models for applications in artificial life, has articulated five basic principles of swarm intelligence:

- The proximity principle: The population should be able to carry out simple space and time computations.
- The quality principle: The population should be able to respond to quality factors in the environment.
- The principle of diverse response: The population should not commit its activity along excessively narrow channels.
- The principle of stability: The population should not change its mode of behaviour every time the environment changes.
- The principle of adaptability: The population must be able to change behaviour mode when it's worth the computational price.

3.2 Particle Swarm Optimization

Particle swarm optimization (PSO) was originally introduced by Eberhart and Kennedy [23, 24, 61]. The PSO algorithm is a population based search algorithm inspired by the social behaviour of bird flocking or fish schooling.

In this algorithm, every individual (*particle*) in the swarm represents a potential candidate solution and it is represented by a multidimensional vector. Each particle has a position vector and a velocity vector. The algorithm determines the rules to update both the velocity and position vectors. The update is conditioned to the best position the entire *population* has visited. The degree of optimality is measured by a fitness function defined by the user. The PSO process then is iterated a fixed number of times or until a minimum error based on desired performance index is achieved.

Although it is a simple model, it has shown great results with difficult optimization problems efficiently. It was originally developed for real valued spaces but nowadays the PSO algorithm has different variants according to the nature of the problem to solve:

- single solution PSO: to find a single solution to unconstrained, static continuous optimization problems;
- niching PSO: to locate more than one solution to an optimization problem;
- constrained PSO: to solve constrained optimization problems;
- multi-objective PSO: to solve problems with multiple conflicting sub-objectives;
- dynamic environment PSO: to locate and track optima in dynamically changing search spaces;
- discrete PSO: to solve problems defined over discrete search spaces.

Since its introduction in 1995, particle swarm optimization has seen many improvements and applications. Most modifications to the basic PSO are directed towards improving convergence of the PSO and increasing the diversity of the swarm. Based on the single solution PSO, there are several variations that modify the basic version to improve the performance, such as the introduction of the velocity clamping or inertia weight.

3.2.1 Basic Particle Swarm Optimization

The following description of the basic PSO and the inertia weight variation can be found at [26]:

> Let $\vec{u}_i(t)$ denote the position of particle i in the search space at time step t; unless otherwise stated, t denotes discrete time steps. The position of the particle is changed by adding a velocity, $\vec{v}_i(t)$, to the current position, i. e.
>
> $$\vec{u}_i(t+1) = \vec{u}_i(t) + \vec{v}_i(t+1) \tag{3.1}$$
>
> with $\vec{u}_i(0) \sim U(\vec{u}_{min}, \vec{u}_{max})$.

The velocity component drives the optimization process weighting the experiential knowledge of the particle (usually called *cognitive component*) and the knowledge of the entire swarm (called *social component*). The *social component* differs depending on the size of neighbourhoods of the particles. There are two main algorithms: the global best algorithm (*gbest*), where there is only one neighbourhood that is the entire swarm; and the *local best* PSO (*lbest*), with several neighbourhoods that can present different topologies.

In the next section we present the Global Best PSO (*gbest*), using the variation with inertia weight.

3.2.1.1 Global Best PSO

For the *gbest* PSO, the neighbourhood for each particle is the entire swarm. The social network employed by the *gbest* PSO is the star topology, that is, each particle has information of all particles in the swarm knowing the best position visited by the entire swarm $\hat{\vec{\mu}}(t)$. $\hat{\vec{\mu}}(t)$ is the best of all best personal positions of all particles.

From [26]:

> For *gbest* PSO, the velocity of a particle i is calculated as:
>
> $$v_{id}(t+1) = v_{id}(t) + k_1 r_{1_d}(t)[\mu_{id}(t) - u_{id}] + k_2 r_{2_d}(t)[\hat{\mu}_d(t) - u_{id}(t)] \tag{3.2}$$
>
> where $v_{id}(t)$ is the velocity of particle i in dimension $d = 1, ..., n_x$ at time step t, $u_{id}(t)$ is the position of particle i in dimension d at time step t, k_1 and k_2 are positive acceleration constants used to scale the contribution of the cognitive and social components respectively, $r_{1_d}(t)$, $r_{2_d}(t) \sim U(0,1)$ are random values in the range [0,1], sampled from a uniform distribution. These random values introduce a stochastic element to the algorithm.
>
> The personal best position, $\vec{\mu}_i(t)$, associated with particle i is the best position the particle has visited since the first time step. Considering minimization problems, the personal best position at the next time step, *t+1*, is calculated as
>
> $$\vec{\mu}_i(t+1) = \begin{cases} \vec{\mu}_i(t) & if\ f(\vec{u}_i(t+1)) \geq f(\vec{\mu}_i(t)) \\ \vec{u}_i(t+1) & if\ f(\vec{u}_i(t+1)) < f(\vec{\mu}_i(t)) \end{cases} \tag{3.3}$$
>
> where $f : \mathbb{R}^{n_x} \to \mathbb{R}$ is the fitness function.

The fitness function measures the quality of the solution found (best particle position), that is, evaluates the success of the optimization process.

From [26]:

> The global best position, $\hat{\vec{\mu}}(t)$, at time step t, is defined as
>
> $$\hat{\vec{\mu}}(t) \in \{\vec{\mu}_1(t), ..., \vec{\mu}_{n_s}(t)\} \,|\, f(\hat{\vec{\mu}}(t)) = min\,\{f(\vec{\mu}_1(t)), ..., f(\vec{\mu}_{n_s}(t))\} \tag{3.4}$$
>
> where n_s is the total number of particles in the swarm.

The gbest PSO is summarized in Algorithm 1, from [26].

Algorithm 1 *gbest* PSO

```
Create and initialize an $n_x-$dimensional swarm;
repeat
  for each particle $i = 1, ..., n_s$ do
    // set the personal best position
    if $f(\vec{u}_i) < f(\vec{\mu}_i)$ then
      $\vec{\mu}_i = \vec{u}_i$
    end if
    //set the global best position
    if $f(\vec{\mu}_i) < f(\hat{\vec{\mu}})$ then
      $\hat{\vec{\mu}} = \vec{\mu}_i$
    end if
  end for
  for each particle $i = 1, ..., n_s$ do
    update the velocity using equation 3.2
    update the position using equation 3.1
  end for
until stopping condition is true;
```

3.2.1.2 The Inertia Weight variation

In an optimization process, a balance between *exploration* (exploring new search space regions) and *exploitation* (in-depth exploring promising regions already found) is required. The inertia coefficient ω was designed to trade-off between this two objectives. It was introduced by Shi and Eberhart [117] as an alternative to *velocity clamping* (another variation to control the global exploration of particles and avoid particles leaving the boundaries of the search space). This coefficient weights the contribution of the previous velocity $\omega v_{id}(t)$.

From [26]:

> For the *gbest* PSO, the velocity equation changes from equation 3.2 to
>
> $$v_{id}(t+1) = \omega v_{id}(t) + k_1 r_{1_d}(t)[\mu_{id}(t) - u_{id}] + k_2 r_{2_d}(t)[\hat{\mu}_d(t) - u_{id}(t)] \tag{3.5}$$

The main disadvantage of using the inertia weight component is that the optimal value of ω is problem dependent. If $\omega<1$, the velocity of particles decreases until zero, facilitating the exploitation. On the contrary, if $\omega>1$ the particles are accelerated to explore new regions. Smaller values of ω involve a major influence of social and cognitive components in the new velocity.

Van den Bergh and Engelbrecht [127] remark the importance of the relation between the value of ω and the values of the acceleration coefficients

k_1, k_2. They stated the following recommendation to ensure convergence of the PSO:

$$\omega > \frac{1}{2}(k_1 + k_2) - 1 \tag{3.6}$$

3.2.2 Implementation Aspects

As algorithm 1 indicates, the optimization process is iterative.

The first step of the PSO algorithm, as usually, is the initialization process. The variables that have to be initialized are:

- The acceleration constants k_1, k_2.
- The inertia weight ω.
- The position of particles. It must be ensured that particles cover all the search space, since these initial positions do influence the efficiency of the algorithm.
- It is recommendable to initialize the velocities to zero, as an analogy with physical objects [26].
- The personal best position for each particle is initialized to the particle's position at time step $t = 0$.

The algorithm ends when a stopping condition is satisfied. There are different options in the literature:

- Terminate when a maximum number of iterations, or FEs (function evaluations), has been reached.
- Terminate when an acceptable solution has been found.
- Terminate when no improvement is observed over a number of iterations.
- Terminate when the normalized swarm radius is close to zero.
- Terminate when the objective function slope is approximately zero.

It is important to select a stopping criterion that does not cause premature convergence of the algorithm. Traditionally, another criterion for selecting the stopping criteria had been minimizing the number of function evaluations, since they increase the computation time; but nowadays this is not such a problem due to the use of parallel computing, as it is supported in this thesis.

In order to take a further look into this subject we do recommend reading [26], where all this information has been extracted from.

3.3 Applications

Scientists have applied these swarm intelligence principles mainly in optimization [23, 24], in control of robots [47, 48], for finding optimal routes [79], scheduling [17], power system controller designs [59], and image [15] and data analysis [77].

In the optimization field, the Swarm is a population of individuals (potential candidate solutions) cooperating among themselves and statistically becoming better and better over time and eventually finding good enough solutions. The main two optimization algorithms based on Swarm Intelligence are Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO).

The PSO algorithm has been applied to many disciplines. For example, in machine learning, to train artificial neural networks [4, 107] and in fuzzy cognitive maps learning [95]. In operations research, it has been applied to scheduling problems [70], and continuous review inventory optimization [96]. It has also been applied to multiobjective optimization [55], minimax optimization [67], constrained optimization [69, 58], etc.

The PSO was originally developed for static environments, but rarely real problems are static. Sometimes the objective function varies over time as a result of the changing environment. Other times, these changes are caused by noise of imprecise information about the variables or the objective function. In the System engineering and automatic control field, a popular control strategy that works successfully with dynamic environments, both in theory and practice, is the Model Based Predictive Control (MBPC). The MBPC uses a model of the system to calculate predictions of the behaviour of the system and optimize the control actions to apply to the system. Since the environment is dynamic, the control action and the predictions are calculated on every time step. The PSO has been successfully applied to dynamic environments [16, 94, 97] (with some changes in the algorithm [26]), and has also been used with the MBPC strategy [60, 134, 8, 132].

In the next chapter...

In the next chapter the process of wastewater treatment is explained and also the activated sludge process. Moreover, some brief fundamentals about microbiology are introduced, which are of interest for understanding the ABM developed at chapter 7.

Chapter 4

Wastewater Treatment

An expert is a person who has made all the mistakes that can be made in a very narrow field.

Niels Bohr

Summary: This chapter encompasses concepts from sewage treatment to more specific concepts of microbiology used in this dissertation.

4.1 Introduction

Water is one of the most fundamental natural resources, and together with air, land and energy are the four bases that support the development. Wastewater or sewage is the polluted water generated by the communities after its use in residential and industrial establishments. Wastewater engineering is a branch of environmental engineering that face the issues associated with the treatment and reuse of wastewater.

The techniques and steps to treat the wastewater are reviewed in section 4.2. Then in section 4.3 the activated sludge process and some of the modelling approaches applied until now are reviewed. After that, in sections 4.4 and 4.5 we face the problem from the point of view of the individuals, that is, microorganisms involved in the process. Some concepts of microbiology are reviewed as well as different modelling approaches (ABM included) under this perspective.

4.2 Wastewater Treatment

The main objective of a wastewater treatment plant (WWTP) is to eliminate the contaminants present in wastewater to facilitate its reuse, avoid the production of malodorous gases or the development of pathogenic microorganisms.

Although there can be some variations, generally wastewater treatment in a WWTP follows the following steps: primary treatment, secondary treatment and tertiary treatment.

Primary Treatment
The objectives of this phase are preparing the influent for the biological treatment (in the secondary treatment) and preventing damage to pumps and other equipment. Large solids present in wastewater are eliminated by screening. Oils, grease and suspended solids are eliminated by means of a combination of flotation, sedimentation, coagulation and filtration processes. The pH of the influent is adjusted if required.

Secondary Treatment
This process removes the organic matter in wastewater by using biological treatment processes, in general, microorganisms that are able to break down and metabolize the main contaminants present in wastewater. In the last stage of this treatment, the microorganisms are allowed to settle down in a clarifier and a fraction of the microorganisms is retired from the effluent to be reused.

Tertiary Treatment
The tertiary treatment aims to generate a higher quality effluent and remove specific types of residuals. Different processes are carried out, such as filtration, to remove suspended or colloidal solids; adsorption and chemical oxidation, to remove organics; disinfection by ozonation or chlorination, etc.

4.3 The Activated Sludge Process

The Activated sludge was first introduced in England in early 1900's and became common in the United States in 1940's.

An activated sludge can be defined as a microbial mass produced when the wastewater is aerated continuously. That mass is made up of microorganisms that are able to break down and metabolize the main contaminants in the wastewater. The activated sludge process is a type of biological process that takes place in the secondary treatment of sewage at a wastewater treatment plant.

Nowadays it is considered as the most important biological wastewater treatment process due to its reliability, flexibility, the high quality effluent it generates, having low construction cost and reasonable operation and maintenance costs, and small land requirement.

The basic activated sludge process, as illustrated on figure 4.1, consists of the following three basic components:

1. An aeration tank (bioreactor) where the activated sludge and wastewater are mixed and aerated for a certain period of time. The microorganisms in the tank are mostly bacteria, although there are other species such as protozoa, fungi, algae, etc.

2. A system of separation of activated sludge and treated effluent (called secondary clarifier or clarifier).

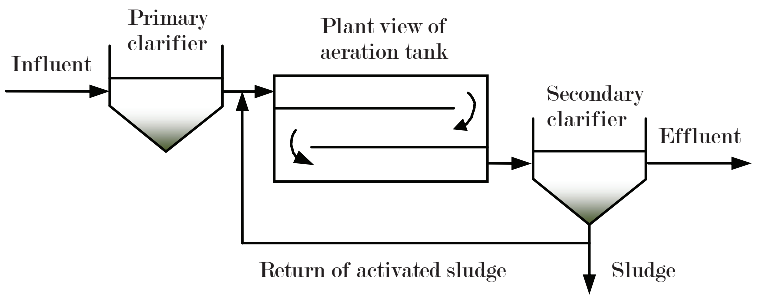
3. A recirculation of activated sludge to the aeration tank.

4. A system of treatment and disposal of the sludge produced (purge).

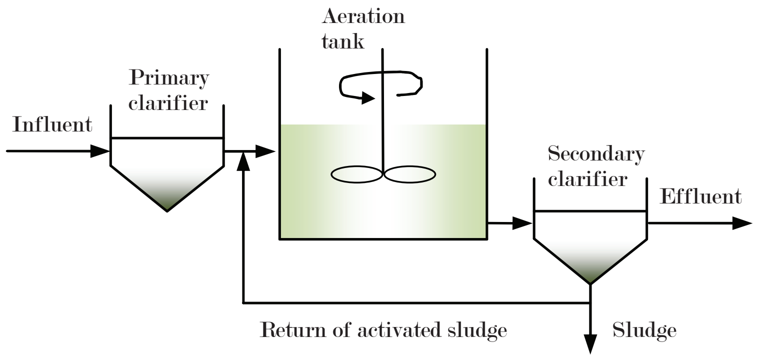
The term *activated* sludge is used because microorganisms reused from the clarifier are in a "hungry" or activated condition, after being for a period of time in the clarifier where there is no substrate.

In this thesis we start modelling a simplified version of the process in a batch reactor, i.e., an isolated reactor in which an initial charge is introduced and provided with adequate ventilation conditions for the optimal development of the reactions of substrate-biomass interaction, in order to better focus on the interaction substrate-biomass.

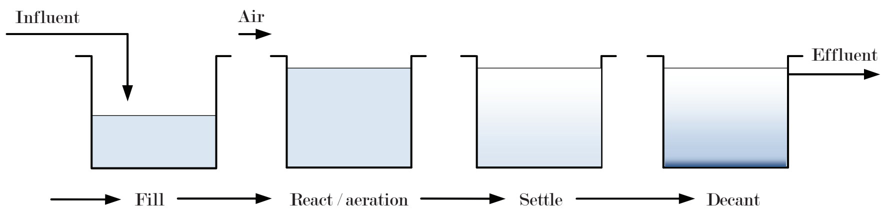
Under these conditions (batch reactor), the temporal evolution of the concentrations of substrate and biomass takes place as shown in Figure 4.2. The growth curve is obtained by a count of the number of living cells (biomass) over time. The biomass curve consists of several phases. The microorganisms must first accommodate to their environment and available food. This accommodation period is called lag phase, and varies in size depending on the history of seeded microorganisms. If microorganisms are adapted to the environment, the lag phase will be very brief. Once growth has begun, it will continue rapidly. When maximum growth is occurring, the behaviour is logarithmic; this is why this phase is called *the logarithmic phase*. Maximum growth cannot continue indefinitely: the food available may be ended up, environmental conditions may change (e.g., overpopulation, accumulation of waste products, etc.), and a population of predators can be developed. Cells that are unable to obtain food from outside sources initiate endogenous catabolism, i.e., they catabolize the protoplasm stored to maintain their energy. Other cells die or break releasing their protoplasm, which is added to the food available. This stage is represented in Figure 4.2 with the name of *stationary phase*, and represents the time along which the production of new cell material is roughly compensated by death and endogenous respiration. Whereas in the stationary phase there is still some reproduction, the



(a) Activate Sludge Reactor. Configuration 1



(b) Activate Sludge Reactor. Configuration 2



(c) Activate Sludge Reactor. Configuration 3

Figure 4.1: Typical activated sludge processes with different types of reactors: (a) schematic flow diagram of plug-flow process and view of plug-flow reactor, (b) schematic flow diagram of a complete-mix process and view of complete-mix activated sludge reactor, and (c) schematic diagram of sequencing batch reactor process and view of sequencing batch reactor (Based on [83])

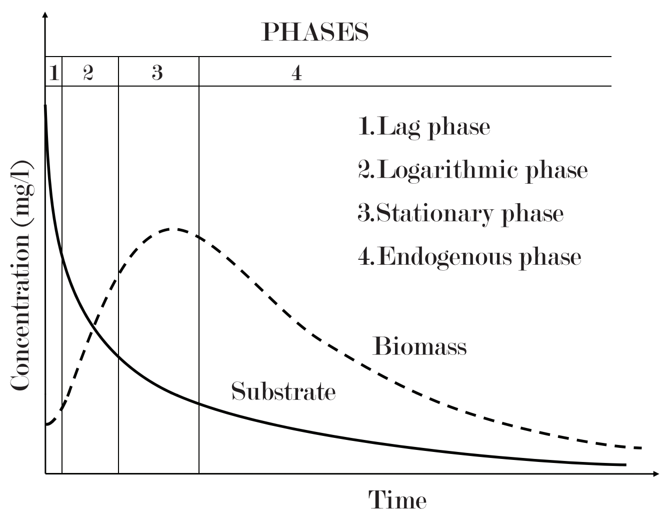


Figure 4.2: Temporal evolution of the concentrations of substrate and biomass.

endogenous respiration and death dominate the fourth phase, called *endogenous phase*. In this last phase, the biomass decreases slowly, asymptotically approaching the horizontal axis.

There are numerous models for the wastewater treatment process, but the most used and generally accepted as a standard in the field of biological modelling of the activated sludge is model ASM1 for the simulation of organic matter removal and biological nitrification and denitrification. Model ASM1 was proposed by the group of mathematical modelling of the International Water Association (IWA). In the book [53], Hence et al. give an overview of the Activated Sludge Model (ASM), presenting ASM1, and the later variations ASM2, ASM2d and ASM3; where the main disadvantage of these models is described as *Practical applicability of these models is limited by the non-availability of activated sludge process data (kinetic coefficients and wastewater characteristics) for a specific wastewater.*

In order to mathematically describe the kinetics of the reactions taking place in a biological reactor, practically all models are based on the two fundamental processes discussed above, that is, microbial growth (μX) and decay ($-K_d X$), usually described mathematically as in equation 4.1:

$$\frac{dX}{dt} = \mu X - K_d X \tag{4.1}$$

where X is the biomass concentration, μ is the specific growth rate and K_d is the decay coefficient. Process stoichiometry is then used to relate substrate (S) utilization to microbial growth, as:

$$\frac{dS}{dt} = -\frac{\mu}{Y}X \tag{4.2}$$

where Y is the yield coefficient, commonly referred to as the substrate-to-biomass yield, which is used to convert between cell growth rate dX/dt and substrate utilization rate dS/dt.

A major issue has been how to mathematically describe the specific growth rate for a continuous culture of microorganisms growing in wastewater on a mixture of organic and inorganic substrates. The most commonly recognized rate expression is the hyperbolic expression proposed by **Monod** (1942; 1949 [87]), as an empirical deduction from pure culture studies, i.e.,

$$\mu = \mu_{max}\frac{S}{K_s + S} \tag{4.3}$$

where μ_{max} is the maximum specific growth rate, K_S is the half-saturation constant (the value of S when $\mu/\mu_{max} = 0.5$), and S is the concentration of the growth-limiting substrate.

The Monod expression is developed as an acceptable mathematical description of experiments conducted with pure bacterial cultures growing on single substrates. In wastewater treatment practice, the biomass concentration is substituted with non-specific parameters like BOD (biological oxygen demand) or COD (chemical oxygen demand). Although they are mathematically treated as single substrate components, these parameters include a great variety of organic compounds with different biodegradation characteristics. The influent wastewater also contains artificially manufactured chemical compounds and toxic materials, to which various organisms respond differently. Furthermore, conditions like the dissolved oxygen (DO) concentration and the pH may vary within the treatment plant. Consequently, in the biological reactors used for the removal of the mixture of organic compounds in wastewaters, there is no way to select a given microbial species, since a mixed microbial community develops as an enriched culture, resulting from natural selection.

4.4 Microbiological concepts

Microorganisms are microscopic organisms consisting of a single cell or cell cluster. Despite of their small size, microorganisms are an essential part of earth. Bacteria represent 50% of the carbon and 90% of the nitrogen and phosphorous of our planet's biomass.

Microbiology is the science that studies microorganisms. It deals not only with the basic biology behind then, but also with many practical problems in medicine, agriculture and industry that are related to microorganisms. Microbial communities are complex systems hardly explained by reduction. Laboratory and field experiments with microorganisms are costly and often unfeasible. Nearly 99% of the known microbial species have not yet been successfully cultured *in vitro*.

Bacteria are unicellular microorganisms. They are procaryotic cells, that is, they lack nuclei, mitochondria and chloroplasts. The procaryotic cells constitute the largest portion of the biomass on Earth. A bacterial culture is the growth of these microorganisms in a culture medium, carried out in a laboratory under controlled conditions. The **batch culture** is a specific kind of culturing in which the bacteria grow in a system of a fixed volume without the addition or removal of the medium. Bacteria divide through binary fission; this results in exponential growth. The growth cycle of a bacterial culture is commonly represented through the **growth curve**, which is the plot of the bacterial concentration over time. It is usually represented on a semilogarithmic plot, in order to identify the exponential growth. We find different phases in the growth curve, depending on the bacteria characteristics and the growth conditions. We may define four phases: lag phase, exponential phase, stationary phase and death phase, as explained at Section 4.3 for the activated sludge process.

The totality of the bacteria of a culture is called **population**. Usually, the term population is used to refer to the total number of cells or the cell density. The **biomass** is the mass of the bacteria, and the total biomass is the mass of the bacteria's population.

These concepts are discussed further in the Thesis of C.P. Soler [105] from which these introductory concepts have been obtained.

4.5 ABM in microbiology

Traditionally, the field of microbiology has always been focused on the description of microbial populations as a whole, adopting a holistic view as opposed to the reductionist view (the properties of a system can be derived from the properties and interrelations of its constituent elements). Until recently, the experimental tools available to microbiology studied bacterial growth on a population scale. The appreciation of the role of individual cells in population dynamics has driven quite some research in the last decades, for a large part because the tools and techniques to study single cells were not available before.

The ABM, or IbM (Individual-based Modelling) as is sometimes known in the microbiology field, can be regarded as an application of the reductionist view. While this view has been routinely applied in other non-biological scientific domains such as particle physics or astronomy, it only started to be applied occasionally in the biological realm in the late 1970s.

Research has clearly shown that individual cell behaviour within a microbial population can be surprisingly diverse, even when the population responses are regular and reproducible. At the lowest level, gene expression is an intensely stochastic process. One of the objectives of current research in the cell population dynamics is the intention to elucidate the **phenotypic cell-to-cell variation** observed in an isogenic cell population even under the same environmental conditions, that is, the intra-population variability than can be produced either by random noises at molecular lever or stochasticities in the cell division process. Another research line is to explore the consequences of **spatial heterogeneity**. Macroscopic approaches often do not consider the principle of *locality*, that is, microorganisms are primarily affected by its spatial-temporal neighbourhood. Macroscopic approaches are mostly used for predictive purposes due to its simplicity and computational efficiency. ABM allows modelling population heterogeneities (intra-population variability) and this has demonstrated to yield different predictions to the traditional macroscopic approaches [116, 44, 52]. *These applications* (ABM) *have revealed potential deficiencies in conventional lumped-typed* (macroscopic) *approaches and provided new insights* [116].

ABM modelling is being increasingly applied in the field of microbiology during the last decade. Examples of application are modelling bacterial colony [38, 65] and biofilm formation [66], and the application in the field of predictive microbiology, the study of prokaryotic cells [38, 65], the study of eukaryotic cells [35, 30], temporal complexity [38, 20, 21], spatial complexity [39], structural complexity [36], etc. The main two ABMs in this field are INDISIM (INDividual DIScrete SIMulations) [38] from the Modelling and Computer Simulation of Biological Systems (MOSIMBIO) group of the *Universitat Politècnica de Catalunya*, and BacSim [65], both modelling bacteria cells.

In relation to wastewater treatment systems, the models of the IWA: ASM1, AMS2 and ASM3 are the current standards, as we mentioned in section 4.3. In ASM2 and ASM3, cell internal storage compounds terms were introduced, which make some scientists to question the validity of such models: *applying kinetic models to the average composition of activated sludge will not necessarily lead to the same model prediction as the sum of all individual behaviors* [44]. ABM has been applied to the study of structures in biolfilms or flocs for the attached-grow processes, and to study the diversity within populations in suspended-grow processes [116, 44, 52]. As Schuler mentions in [116], the future research lines are in modelling evolution/adaptation of bacteria in wastewater treatment systems, validation of heterogeneity predictions of ABM models, the combined use of ABM and computational fluid dynamics, etc. Currently the two main ABM models are the one proposed by Gujer [44] for continuous and batch suspended growth systems, and the DisSimulator IBM from Schuler [115] which is more complete and complex than the one from Gujer.

In the next chapter...

In the next section we review some concepts about general purpose programming with graphics processing units (GPUs).

Chapter 5

Parallel Computing on the GPU with CUDA

The world is round and the place which may seem like the end may also be the beginning.

Ivy Baker Priest

Summary: In this chapter some concepts about parallel computing using the power of graphic cards are going to be reviewed, as well as some fundamentals about the programming architecture from NVIDIA cards, the so called CUDA.

5.1 Introduction. Parallel computing revolution has already happened.

Historically, the method to increase the performance of computing devices has been to speed up the processor's clock. First computers had central processing units (CPUs) with internal clocks operating around 1MHz. Nowadays, 30 years later, these clock speeds range between 1GHz and 4GHz. As it is said in [19]: "one of the problems with today's modern processors is they have hit a clock rate limit at around 4 GHz. At this point they just generate too much heat for the current technology and require special and expensive cooling solutions. This is because as we increase the clock rate, the power consumption rises. In fact, the power consumption of a CPU, if the voltage is fixed, is approximately the cube of its clock rate. To make things worse, as the heat generated by the CPU is increased, for the same clock rate, the power consumption also increases due to the properties of the silicon. This conversion of power into heat is a complete waste of energy. This increasingly inefficient use of power eventually means you are unable to either power or cool the processor sufficiently and you reach the thermal limits of the device or its housing, the so-called "power wall". In 2005, the leading CPU manufacturers began offering processors with two computing cores instead of one, to increase the computing power of new computers.

In the late 90s, computer scientists from various fields started using GPUs to accelerate a range of scientific applications. They tried to take advantage of the computing power of a device made up of hundreds of cores. This movement was called GPGPU (General-Purpose computation on GPU). It was not an easy thing to do; scientists had to manage to code their programs into pixels and vertex operations, using specific tools.

NVIDIA recognized the potential of bringing this performance for the larger scientific community and, in November 2006, NVIDIA introduced CUDA (Compute Unified Device Architecture), a general purpose parallel computing architecture, with a new parallel programming model and instruction set architecture. The GeForce 8800 GTX was the first GPU to be built with NVIDIA's CUDA architecture. This architecture included several new components designed strictly for GPU computing and aimed to alleviate many of the limitations that prevented previous graphics processors from being legitimately useful for general-purpose computation [113]. In 2010, a NVIDIA GPU-based machine was listed as the second most powerful computer in the world, according to the top 500 list (http://www.top500.org). In 2011, NVIDIA CUDA-powered GPUs went on to claim the title of the fastest supercomputer in the world [19].

The most widely used parallel programming languages today are Message Passing Interface (MPI) for scalable cluster computing, OpenMP for shared-memory multiprocessor systems, CUDA for the GPU computing on NVIDIA cards and OpenCL, standardized programming model developed between the major manufacturers (Intel, AMD/ATI, and NVIDIA).

5.2 What is the CUDA architecture

CUDA is a parallel computing platform and programming model that enables dramatic increases in computing performance by harnessing the power of the graphics processing unit (GPU) [91]

The CUDA architecture was a revolution in the technique used to program GPUs, outstripping the need to make the problem to look like a computer graphics task.

NVIDIA provided their new family of GPUs with ALUs (Arithmetic Logic Units) that complied with IEEE requirements for single-precision floating-point arithmetic in order to be used for general-purpose computations.

NVIDIA also developed a language, CUDA C, the first language specifically designed by a GPU company to facilitate general-purpose computing on GPU. CUDA C adds a set of instructions and keywords to the standard ANSI C, reducing the learning curve for C programmers. NVIDIA also provides a specialized hardware driver to exploit the CUDA architecture.

Nowadays, a lot of applications are benefited from the CUDA speed up, from medical imaging, to computational fluid dynamics, DNA sequencing, etc.

5.3 CUDA program structure

A CUDA program consists of different parts that are executed sequentially on either the *host* (the CPU) or in the *device* (a compatible GPU). The code parts that exhibit little or no data parallelism are implemented in host code and the ones that can be benefited of the parallelism of code are implemented in device code. So, a CUDA program encompasses both host and device code. The host code is straight ANSI C.

The device code follows the SIMD (Single Instruction, Multiple Data) computing model, where the same instructions are massively executed over a huge amount of data. The device code consists of *kernel* functions (or simply *kernels*) that typically generate a large number of threads to exploit data parallelism. All the threads generated by a kernel during an invocation are collectively called a *grid*, and also this grid is divided in *blocks*. All blocks in a grid have the same number of threads . To distinguish between threads, they are identified with a code of coordinates (usually called *index*) composed by the block identifier (block index within the grid) and the thread identifier (thread index within the block). For example, for a 1-D dimensional block we only use the x coordinate of the block and thread, composing the thread global identificator as: *index = blockIdx.x * blockDim.x + threadIdx.x.* The thread organization in a CUDA two dimensional grid is shown at Figure 5.1.

The execution of a CUDA program starts with host code, where memory allocation and initializations are done. Then, the data is transferred from host memory to device memory. Afterwards, the device functions are invoked and a large number of threads are launched (the number of threads and its configuration is defined in the invocation code), as it is illustrated in Figure 5.2. At last, the data is transferred back to the host.

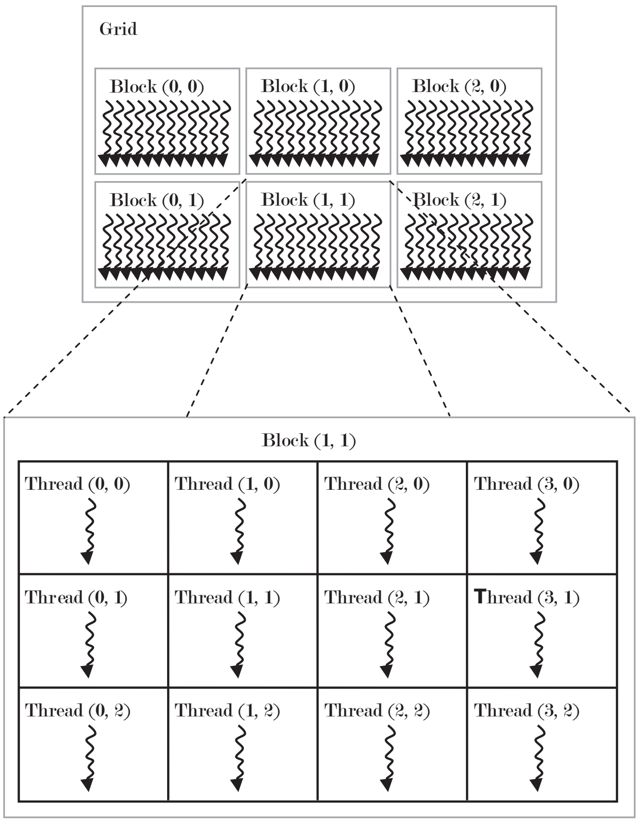


Figure 5.1: Thread organization. Based on [91].

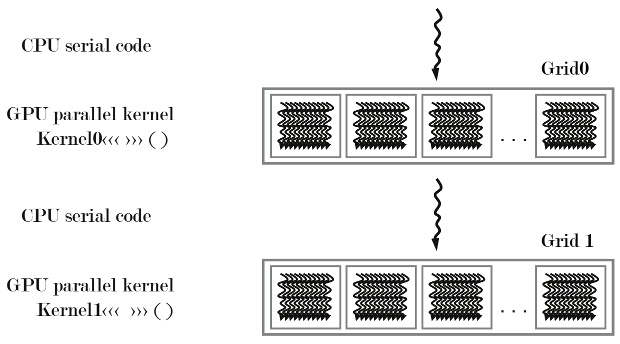


Figure 5.2: Execution of a CUDA program. Based on [62].

5.4 The NVIDIA GPU hardware

In general, a GPU is a device composed by hundreds of computational cores, grouped in arrays that share some memory spaces. The computational cores are called streaming processors (SPs), and they are grouped forming streaming multiprocessors (SMs). The number of SPs in each SM depends on the GPU model, and it ranges from 8 to 192 SPs.

There are different types of memory in a GPU. The global memory is readable and writeable by all threads, and is the one that present longer latencies because it is implemented with dynamic random access memory (DRAM). Each block shares some memory faster than the global one (called *shared memory*) and contains some registers (each thread can only access its own registers). There is also some memory spaces only readable called *constant memory* and *texture memory*, with faster access. Figure 5.3 is a visual summary of this memory structure.

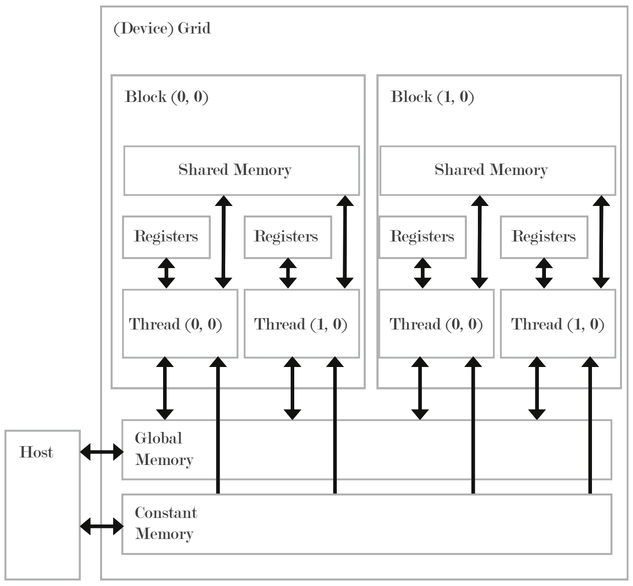


Figure 5.3: Memory structure of a NVIDIA GPU. Based on [62].

5.5 Example of a CUDA C program

In this section we present a simple example code in CUDA. The example allocates an array and performs a simple operation on every element in the array: *element = 2 * index-of-the-element-in-the-array*, and prints the result on the screen.

The standard C code for this example is:

```
#include <stdlib.h>
#include <stdio.h>

// Function 2*i
void 2timesi (int n, int *array)
{
for(int i=0; i<n; ++i)
array[i]= 2*i;
}


//Main function
int main(void)
{
int num_elements = 2560;
int num_bytes = num_elements * sizeof(int);

//pointer to the array
int* host_array=NULL;

//malloc the array
host_array = (int*)malloc(num_bytes);

//perform 2timesi on num_elements elements
2timesi(num_elements, *host_array)

// print out the result element by element
  for(int i=0; i < num_elements; ++i)
  {
    printf("\%d ", host_array[i]);
  }

  // deallocate memory
  free(host_array);
}
```

The CUDA C code for this example is:

```
#include <stdlib.h>
#include <stdio.h>

__global__ void twotimesi(int *array)
```

```
{
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  array[i]= 2*i;
}

int main(void)
{
  int num_elements = 2560;
  int num_bytes = num_elements * sizeof(int);

  // pointers to host & device arrays
  int *device_array = 0;
  int *host_array = 0;

  // malloc a host array
  host_array = (int*)malloc(num_bytes);

  // cudaMalloc a device array
  cudaMalloc((void**)&device_array, num_bytes);

// Execution configuration
  int block_size = num_elements;
  int grid_size = 1;

//perform twotimesi on num_elements elements
  twotimesi<<<grid_size,block_size>>>(device_array);

  // download and inspect the result on the host:
  cudaMemcpy(host_array, device_array, num_bytes, cudaMemcpyDeviceToHost);

  // print out the result element by element
  for(int i=0; i < num_elements; ++i)
  {
    printf("%d ", host_array[i]);
  }

  // deallocate memory
  free(host_array);
  cudaFree(device_array);
}
```

The differences between both implementations are:

- The CUDA code needs two allocations instead of one, one for the GPU memory and one for the CPU memory.

- The execution configuration determines the size of the grid (one block in this example) and the size of the block (*num_elements* threads).
- The operation on every element of the array is performed in parallel in the CUDA example.
- In the CUDA code, data has to be transferred back to the CPU memory if you want to have it available for other calculus or visualization.

The operation *twotimesi* is performed over 2560 values (one for each array element), with a duration of 0.017 seconds for the ANSI C code, and 0.005 for the CUDA C code. The acceleration for this simple example is 3.4x.

5.6 Choosing a NVIDIA GPU for computing

NVIDIA has three families of GPUs that support GPU computing. *GeForce* family is designed for consumer graphics, *Quadro* is designed for professional visualization, and *Tesla* is designed for parallel computing and programming and offers exclusive high performance computing features. The performance of the GPU is consistent with the price; *GeForce* are the cheapest, and *Tesla* is a investment to value.

Tesla architecture was released in 2007, *Fermi* in 2009, and *Kepler* in 2012. The GPU architecture implies different features and also imposes different programming limitations [71]. For example, *Kepler* allows dynamic allocation of memory (not allowed in *Fermi* nor *Tesla*); *Fermi* increased the number of SPs per SM from the previous *Tesla* architecture, and *Kepler* also increased them over *Fermi*'s; in *Fermi*, data could only be exchanged between threads using shared memory (resulted in additional synchronization time); *Kepler* allows the *shuffle functions* to exchange data between threads without using shared memory; etc.

The GPUs are also grouped by *compute capability* [91] and by architecture (*Tesla, Fermi, Kepler*):

> The *compute capability* of a device is defined by a major revision number and a minor revision number. Devices with the same major revision number are of the same core architecture. The major revision number is 3 for devices based on the Kepler architecture, 2 for devices based on the Fermi architecture, and 1 for devices based on the Tesla architecture. The minor revision number corresponds to an incremental improvement to the core architecture, possibly including new features.

In this dissertation we have used a *GeForce GTX 480*, with *Fermi* architecture.

5.7 The Parallel Computing Toolbox from MATLAB

The *Parallel Computing Toolbox* from MATLAB allows to execute parallel code in the CPU cores and also in a compatible CUDA GPU.

For GPU computing, it provides a set of functions to be used directly in the MATLAB command line and that are executed in parallel. You can allocate an array, evaluate a function on each element of an array, etc. Although it has some limitations, it is very easy to use and a good starting point. It is under continuous development, and nowadays the MATLAB developers team has joined the Accelereyes team (that offered a similar product years ago [3]) to develop a unified and improved product for GPU computing in MATLAB.

The supported GPUs are NVIDIA CUDA GPUs with compute capability version 1.3 or higher. These GPUs support double-precision computations.

5.8 Swarm intelligence and ABM applications of GPU computing

GPU computing has been recently used to accelerate swarm intelligence and ABM applications. There are only a few works on the subject. A framework for developing ABM with CUDA has been presented recently (2013); it is called FLAME GPU [111] and is an extension to the FLAME framework (Flexible Large-scale Agent Modelling Environment) [31], a generic agent-based modelling system. Wenwu Tang has presented this year (2013) a set of advices for ABM parallel modelling [121]. In the literature we can find some works accelerating ABMs with GPU [1, 74, 122] and many works implementing the Particle Swarm Optimization algorithm [90, 120, 133].

Part III

Developments

In this part, the development work carried out in this thesis is explained in detail.

Chapter 6

The State Space Agent-Based Representation

The best scientist is open to experience and begins with romance - the idea that anything is possible.

Ray Bradbury.

Summary: In this chapter we introduce a representation for agent-based models, based on the state space representation of systems commonly used in system engineering, that can be applied as a complement to the ODD protocol.

6.1 Introduction

The aim of this representation SSABR (State Space Agent-Based Representation) is to bring the methodology of ABM closer to the System Engineering and Automatic Control field. It is based on the state space representation and also in the black box representation of systems. Our objective is not to substitute the current ODD protocol [41, 42, 43], but to provide a representation based on the systems engineering approach. In this sense, it can be considered as a complement to the ODD protocol, focused on the state dynamics description.

The proposed notation (SSABR) is composed of a mathematical description of the dynamic states of the agents, a set of functional relationships between states and parameters leading to the model dynamics, and a formulation of output functions that leads to the measured aggregated variables of the model.

6.2 Mathematical Description

An ABM is composed of a population of agents (grouped into types or 'breeds') or by agents all of the same type. If there are different types, these are identified by the superindex j, which ranges from 1 to the total number of breeds (b).

Each agent i_j of type j can be represented by a vector of its dynamic states $\vec{x}^j_{i_j}(t)$ at time t, and all agents of the same type j are grouped into a matrix of states vectors $X^j(t)$.

In some models, the number of agents n_j can change during the simulation; this is going to be expressed as a time dependency: $n_j(t)$. If the number of agents is constant, this time dependency can be suppressed.

Each agent may also have some parameters $\vec{p}^j_{i_j}$ that influence its dynamics.

In summary, the **State of an agent** at time t is represented by $\vec{x}^j_{i_j}(t)$, with each component of the vector being the value of a dynamic attribute of the agent.

$j = 1, ..., b$ identifies the type of the agent ('breed')

$i_j = 1, ..., n_j(t)$ identifies the agent of type j

$n_j(t)$ is the number of agents of type j at time t

$\vec{x}^j_{i_j}(t) \in \mathbb{R}^{m_j}$, m_j is the number of states for type j

$\vec{p}^j_{i_j} \in \mathbb{R}^{q_j}$, q_j is the number of parameters for type j

State of the set of agents j

$X^j(t) = \left(\vec{x}^j_1(t) \ ... \ \vec{x}^j_{i_{n_j(t)}}(t)\right)$, $X^j(t) \in M_{m_j \times n_j(t)}(\mathbb{R})$

Parameters of the set of agents j

$P^j = \left(\vec{p}^j_1 \ ... \ \vec{p}^j_{i_{n_j(t)}}\right)$, $P^j \in M_{q_j \times n_j(t)}(\mathbb{R})$

Parameters of the environment

$\vec{\alpha} \in \mathbb{S} \subset \mathbb{R}^l$, where l is the number of parameters related with the environment.

From now on, for notation simplicity, we drop the dependency of $\vec{x}^j_{i_j}$ on time t, that is, $\vec{x}^j_{i_j} \equiv \vec{x}^j_{i_j}(t)$

Dynamic methods

An agent i_j of type j has an interaction 'a' with other agents and/or with the environment. This interaction is expressed by means of a method 'a' that depends on the states and parameters of all the agents (in general) and of the parameters of the environment. This interaction causes the states of some of the agents to change. The methods can also represent a consequence of time passing, instead of representing an interaction; for example, the increment on age of an agent.

It is not trivial to express the agent interactions in mathematical formulation due to the fact that the interactions of some agents can affect other agents.

We are only interested in the functional relationships between states and parameters that arise from these interactions. If a complete description of the methods is required, one would have to resort to an algorithmic description or using more formal frameworks, such as language Z [104].

The evolution of the model at time t follows three steps. First, a copy of the states matrix (Z^j) of the agents and of the number of agents (c_j) at time t is made (step 1). Then (step 2), the states are evolved applying the agent methods (see 2.5.2), that is, the agent states are updated ($F^j_{i_j,a}$ is the method representing the interaction 'a' performed by the agent i_j of type j). This is performed over all types of agents, over all agents in the agent set, and for every method. At last (step 3), once all methods at time t have been applied, the states at time t+1 and the number of agents are updated based on the copies.

Step 1:

$$\left\{Z^j = X^j(t);\;\; c_j = n_j(t);\;\; j = 1,\ ...,\ b\right\}.$$

Step 2:

$$\left\{\begin{array}{l} \left\{\bar{Z}^k;\ k=1,\ ...,\ b\right\} = F^j_{i_j,a}\left(\left\{Z^k;\ k=1,\ ...,\ b\right\},\ \left\{P^k;\ k=1,\ ...,\ b\right\},\right. \\ \left.\vec{\alpha}\right) \\ \bar{Z}^k \in M_{m_k\times\bar{c}_k}(\mathbb{R});\ \bar{c}_k \in \mathbb{N}\ \textit{new number of agents of type } k \\ \textit{Substitute } Z^k \textit{ by } \bar{Z}^k \textit{ and } c_k \textit{ by } \bar{c}_k;\ k=1,\ ...,\ b \end{array}\right.$$

$$j = 1,\ ...,\ b$$
$$i_j = 1,\ ...,\ c_j$$
$$a = 1,\ ...,\ A \qquad A \in \mathbb{N}\text{: number of interactions}$$

Step 3:

$$\left\{X^j(t+1) = \bar{Z}^j;\;\; n_j(t+1) = \bar{c}_j;\;\; j = 1,\ ...,\ b\right\}.$$

Aggregated variables

At each step of the simulation, the aggregated variables $\vec{y}(t)$ summarizing the global behaviour of the model are computed. These variables are expressed by means of relationships between the agent states and the parameters of the agents and of the environment. This functional relationship at time t can encompass past values of the states, parameters and even the previous value of the aggregated variable (such as in an accumulator).

$\vec{y}(t) = \vec{g}(\vec{y}(t-1),\ \left\{X^j(\tau);\ j = 1,\ ...,\ b;\ \tau = 0,\ ...,\ t\right\},$
$\left\{P^j;\ j = 1,\ ...,\ b\right\},\ \vec{\alpha})$

$\vec{y}(t) \in \mathbb{R}^S$, S: number of aggregated variables

6.3 Examples

In this section we present a collection of simple ABM examples described using the notation SSABR proposed in this work. Some comments are added to clarify some particularities that can appear in the examples.

The first example is exposed at the subsection 6.3.1; the rest of them can be developed in a similar way and is collected in the appendix A.

6.3.1 Segregation model

This model [128] comes from the Netlogo [129] library of examples. It is a version of the *Thomas Schelling segregation model*, also known as *Schelling tipping model* [114].

In the 1970s, Thomas Schelling proposed a simple population model to illustrate how, even with minimal assumptions about individuals' neighbour preferences, an integrated city would evolve to a segregated city, even if all individuals prefer integration. Schelling placed pennies and dimes on a chess board, as agents representing any two groups in society, and moved them around according to various rules. He interpreted the board as a city, with each square of the board representing a house or a lot. The neighbourhood of an agent occupying any location on the board consisted of the squares adjacent to this location. Rules could be specified that determined whether a particular agent was happy in its current location. If it was unhappy, it would try to move to another location on the board. This model is one of the firsts agent based models in the social science.

In the Netlogo implementation, there are two groups of agents, red turtles and green turtles. Using the SSABR, we are going to define one type of agents, the turtles, with three states: its coordinates in a two dimensional grid environment (x_cell, y_cell) and a happiness state ($happy?$) as table 6.1 shows. To separate the turtles into two categories, red and green, we use a parameter *colour*. They both have a percentage of similar neighbours that make them feel happy, expressed by the parameter $\% - similar - wanted$ (table 6.2).

Applying SSABR will lead to the following

Turtles

$j = 1$, turtle agent type

$$\begin{aligned}
&\vec{x}^1_{i_1} \in \mathbb{R}^3 \; i_1 = 1, ..., n_1 \\
&\vec{x}^1_{i_1} = \begin{pmatrix} x\ cell \\ y\ cell \\ happy? \end{pmatrix} \\
&X^1(t) = \left(\vec{x}^1_1 \; ... \; \vec{x}^1_{n_1}\right); \; X^1(t) \in M_{3\times n_1}(\mathbb{R})
\end{aligned}$$

We are going to make some partitions $(x^1_1(t), x^1_2(t), x^1_3(t))$ from $X^1(t)$ for convenience, that allows to select each state for all the agents in a separate way.

$$X^1(t) = \begin{pmatrix} x^1_1(t) \\ x^1_2(t) \\ x^1_3(t) \end{pmatrix}; \; x^1_1(t), \; ..., \; x^1_3(t) \in M_{1\times n_1}(\mathbb{R})$$

$$\begin{aligned}
&\vec{p}^1_{i_1} \in \mathbb{R}^2 \\
&\vec{p}^1_{i_1} = \begin{pmatrix} colour \\ \% - similar - wanted \end{pmatrix} \\
&P^1 = \left(\vec{p}^1_1 \; ... \; \vec{p}^1_{n_1}\right); \; P^1 \in M_{2\times n_1}(\mathbb{R})
\end{aligned}$$

Agent type	State	Description
Turtle	*x cell*	x coordinate of the cell where the agent is located in the environment
Turtle	*y cell*	y coordinate of the cell where the agent is located in the environment
Turtle	*happy*?	(boolean) true if %-similar-wanted of that turtles' neighbours are the same colour as itself

Table 6.1: Turtle states

Agent type	Parameter	Description
Turtle	*colour*	turtle colour can be red or green
Turtle	$\% - similar - wanted$	percentage of same-colour turtles that each turtle wants among its neighbours

Table 6.2: Parameters

The environment

$\vec{\alpha} \in \mathbb{R}^2$

$$\vec{\alpha} = \begin{pmatrix} number - of - width - cells \\ number - of - height - cells \end{pmatrix}$$

Dynamic methods

There is only one method ($F^1_{i_1}$) for each agent at time t. This project models the behaviour of two types of turtles in a mythical pond. The red turtles and green turtles get along with one another. But each turtle wants to make sure that it lives near ($z^1_1,\ z^1_2$) some of 'its own' (p^1_1), that is, each red turtle wants to live near at least some red turtles, and each green turtle wants to live near at least some (p^1_2) green turtles. If turtles don't have enough (p^1_2) same-colour (p^1_1) neighbours, they are unhappy (z^1_3), and they jump to a nearby patch ($\bar{z}^1_1,\ \bar{z}^1_2$) inside the environment limits ($\vec{\alpha}$). If they are happy (z^1_3), they do nothing. The simulation stops when all turtles are happy.

Step 1:

$Z^1 = X^1(t) \quad c_1 = n_1$

Step 2:

$$Z^1 = \begin{pmatrix} z^1_1 \\ z^1_2 \\ z^1_3 \end{pmatrix} \quad \bar{Z}^1 = \begin{pmatrix} \bar{z}^1_1 \\ \bar{z}^1_2 \\ \bar{z}^1_3 \end{pmatrix}$$

$z^1_1,\ ...,\ z^1_3 \in M_{1\times c_1}(\mathbb{R});\ \bar{z}^1_1,\ ...,\ \bar{z}^1_3 \in M_{1\times \bar{c}_1}(\mathbb{R})$

Partitions in Z^1 and $\bar{Z}^1$ for convenience, as in the definition of turtle agent states.

$$P^1 = \begin{pmatrix} p^1_1 \\ p^1_2 \end{pmatrix}$$

$p^1_1,\ ...,\ p^1_2 \in M_{1\times c_1}(\mathbb{R})$ Partitions for convenience

$$\begin{pmatrix} \bar{z}^1_1 \\ \bar{z}^1_2 \\ \bar{z}^1_3 \end{pmatrix} = F^1_{i_1}(z^1_1,\ z^1_2,\ z^1_3,\ p^1_1,\ p^1_2,\ \vec{\alpha}) \qquad i_1 = 1, ..., c1$$

Step 3:

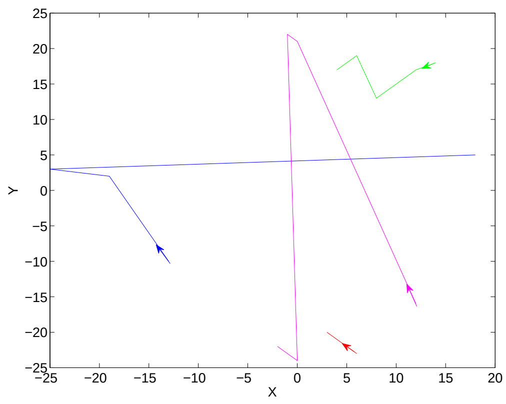
$X^1(t+1) = \bar{Z}^1 \quad n_1 = \bar{c}_1$

Aggregated variables

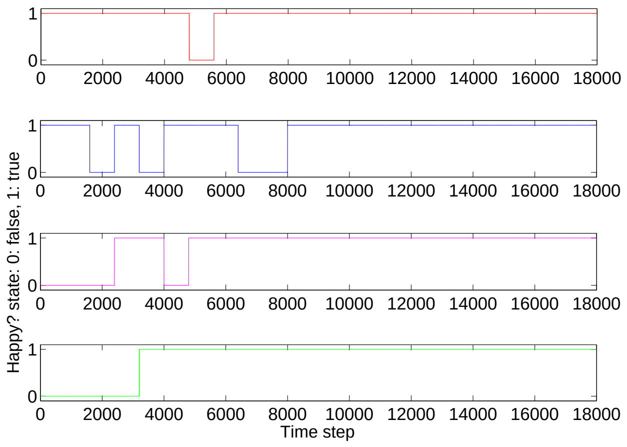
In this model there are two aggregated variables computed, so $\vec{y}(t)$ has two dimensions ($\vec{y}(t) \in \mathbb{R}^2$).

The first aggregated variable $y_1(t)$ is the mean of the percentage of similar colour in the neighbourhood of each turtle.

$y_1(t) = g_1(x^1_1(t),\ x^1_2(t),\ p^1_1)$



(a) Evolution of position states of four turtles.



(b) Evolution of *happy?* states of four turtles.

Figure 6.1: Evolution of turtle states.

The second one, $y_2(t)$, is the percentage of unhappy turtles referred to the total number of turtles.

$y_2(t) = g_2(x_3^1)$

6.3.1.1 Simulation example

We simulated the model with 800 turtles (400 green turtles and 400 red turtles) and $\% - similar - wanted$=62%. The dynamic evolution of the states for four turtles is shown in figures 6.1a and 6.1b.

6.4 Conclusions

A notation SSABR for representing agent-based models has been proposed, which is closer to the state space representation in the systems engineering field and points out the dynamic nature of agents' behaviour through a state space representation.

In the next chapter...

In the next chapter we present our agent-based model for the activated sludge process.

Chapter 7

Agent-Based Model of the Activated Sludge Process

All models are wrong, but some are useful.

George E. P. Box, 1979.

Summary: In this chapter we present a model for the activated sludge process. We summarize some steps of the modelling process and we present three versions, from the preliminary earlier version to the refined final one. We also present some of the experiments conducted to test the models and some interesting results are presented at the end.

7.1 Introduction

As it was stated before at section 1.2, the objective of this part is to apply the agent-based modelling approach to a system typically used in the process system engineering field, the activated sludge process.

The early stages of the modelling process needed some simplifications, so some assumptions were made:

- We model the reactor, without the clarifier.
- In the first version of the model, we model the reactor in batch operation protocol, explained in section 4.3.
- A single population of bacteria is modelled, since the majority of microorganisms in an activated sludge process are bacteria.

- We do not model the aeration system. The tank is supposed to have such systems that provide the necessary oxygen for aerobic bacterial action, and that prevents the sludge from settling providing its homogenisation.

The next step in the modelling process was the selection of the modelling environment. The first software environment tested was NetLogo. It is a great platform for beginners since its interface is easy to use, there are a huge number of examples of models in the program (*Models Library*), the documentation that it is distributed with the program is complete and easy-to-use, and it is a well known environment in the field. This first version of the model is explained at section 7.2.

After that, we decided to change the modelling environment to have more control in the programming process and the possibility to use a variety of toolboxes when using the model. MATLAB was chosen because it is a high level scientific and engineering computational tool, a lot of specialized toolboxes can be incorporated, a custom graphic user interface can be easily built so non-technical users can use it, it provides versatility and integration with other tools (such as CUDA, optimization algorithms, neural network toolbox...), and also because it has been used traditionally in the process and system engineering field. This modelling environment offers a lot of possibilities for modelling agents. We explored the different options for representing agents and finally selected the vectorial programming paradigm. This is exposed at section 7.3.

Once we had chosen the programming alternative in MATLAB, we decided to eliminate one of the simplifications and construct a benchmark that allows proving different operation protocols for the activated sludge model, that is, a continuous flow process, a semi-batch process, and the previous batch process. This is explained in more detail in section 7.4.

7.2 Activated Sludge Batch Reactor Model. NetLogo version

The different versions of the model are described in this chapter with the ODD protocol (section 2.7), since it is the currently most accepted standard for presenting an ABM.

7.2.1 Description

7.2.1.1 Overview

Purpose. The purpose of the model is to simulate the behaviour of a population of bacteria in an activated sludge batch reactor to be use as an alternative to the traditional equation-based models. The ABM technique results in an adequate approach to model bacterial population variability that is not possible with the traditional models. Moreover, this type of models allows a better understanding of the underlying process and interactions between the components of the systems.

Entities, state variables, and scales. The basic entities of the model are bacteria, substrate units and spatial cells. Each bacterium is represented with four variables: location (spatial cell where they are), size, age, and internal energy. The substrate units are represented by agents with the state variables: location and size. The spatial cells have two coordinates: x and y (the environment is two-dimensional). This is a non-parametrized version so there is no scale to make a correspondence with real units. The parameters *b_density* and *s_density* are scale factors in order to compare reactors of different sizes; by default their value are set to 1 (no influence).

Process overview and scheduling. At each time step of the simulation the following actions are performed: to shake, to eat, to reproduce, to die, to update internal energy. The process of updating the state variables is asynchronous (the asynchronous schedule was explained in subsection 2.5.3).

7.2.1.2 Design concepts

Emergence. The dynamics of the system are the result of the interaction between bacteria and substrate. The structure of the bacterial population (parametrized in the initialization process) determines the dynamics.

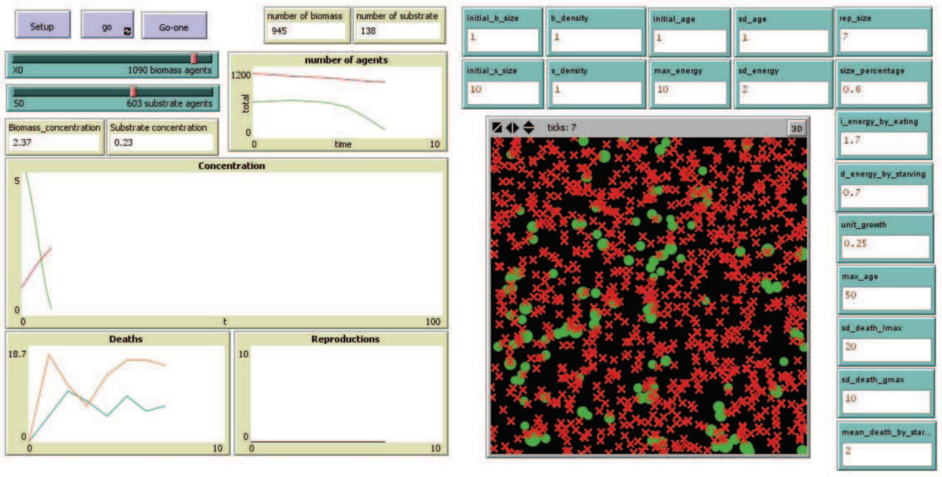
Adaptation. Bacteria reduce their internal energy (reduce their stored protoplasm) when there is a lack of nutrient.

Sensing. Bacteria perceive the presence of substrate in their surroundings.

Interaction. Bacteria directly interact with the substrate metabolising it. Interactions between bacteria are indirect as a result of the interaction between bacteria and substrate. There is no competition between bacteria.

Stochasticity. Stochasticity is used to provide the bacterial population with heterogeneity characteristics, initializing their characteristics around mean values, using a normal distribution (the most commonly encountered distribution in nature). It is also used when applying the rules that govern the model, modelling the stochasticity that is present in nature. Also, the order in which the individuals perform the actions is random, avoiding privileging first-acting consequences (NetLogo intrinsically provides this characteristic).

Observation. Data collected from the model corresponds to the evolution of the following population variables: number of bacteria, number of substrate units, concentration of bacterial biomass, concentration of substrate, reproductions and deaths (Figure 7.1).



number of biomass 945
number of substrate 138
X0 1090 biomass agents
S0 603 substrate agents
Biomass_concentration 2.37
Substrate concentration 0.23
initial_b_size 1
b_density 1
initial_age 1
sd_age 1
rep_size 7
initial_s_size 10
s_density 1
max_energy 10
sd_energy 2
size_percentage 0.6
i_energy_by_eating 1.7
d_energy_by_starving 0.7
unit_growth 0.25
max_age 50
sd_death_lmax 20
sd_death_gmax 10
mean_death_by_star... 2

Figure 7.1: Screenshot of the Netlogo model interface.

7.2.1.3 Details

Initialization. The initialization process is conducted by the user when pressing the *setup* button. The initialization of the model is as follows: read configuration parameter values from the interface, assign values to the constants of the model, create the agents and place them in the world, and assign initial values of agent variable states. Initially, agents are placed randomly on the environment as a uniform distribution. Bacterium agents are created with initial energy values as a normal distribution $N(max\_energy, sd\_energy)$, initial age values as a normal distribution $N(initial\_age, sd\_age)$, and constant size $initial\_b\_size$. Substrate agents are created with constant size $initial\_s\_size$. The inputs that have to be set up by the user in the interface are: *number of bacterium agents*, *number of substrate agents*, $initial\_b\_size$, $initial\_s\_size$, $b\_density$, $s\_density$, $initial\_age$, sd_age, max_energy, sd_energy, rep_size, $size\_percentage$, $i\_energy\_by\_eating$, $d\_energy\_by\_starving$, $unit\_growth$, max_age, sd_death_lmax, sd_death_gmax, $mean\_death\_by\_starving$. These parameters are explained in table 7.1.

Submodels.

- To shake. At each step of the program, agents in the world are rearranged in a random position, thus modelling a system that operates in continuous stirred-tank reactor (CSTR) conditions, obtaining homogeneous characteristics in terms of substrate and biomass concentrations.

Initialization parameter	Value
$number\ of\ bacterium\ agents$	Number of bacterium agents inside the reactor at time step zero.
$number\ of\ substrate\ agents$	Number of substrate agents inside the reactor at time step zero.
$initial\_b\_size$	Mean of bacterial initial size value.
$initial\_s\_size$	Mean of substrate initial size value.
$initial\_age$	Mean of bacterial initial age value.
sd_age	Standard Deviation of bacterial initial age value.
sd_energy	Standard Deviation for energy at birth.
Model parameter	Value
$b\_density$	Bacteria scale factor.
$s\_density$	Substrate scale factor.
max_energy	Maximum energy value that a bacterium agent can reach.
rep_size	Size from which bacteria are able to reproduce.
$size\_percentage$	Decrease of the size of a substrate agent eaten by a bacterium agent.
$i\_energy\_by\_eating$	Increase of energy of a bacterium agent by eating.
$d\_energy\_by\_starving$	Decrease of energy of a bacterium agent every iteration.
$unit\_growth$	Amount by which the size of a bacterium agent is increased when eating.
max_age	Maximum age that a bacterium agent can reach.
sd_death_lmax	Standard deviation for age deaths when a bacterium agent is younger than max_age.
sd_death_gmax	Standard deviation for age deaths when a bacterium agent is older than max_age
$mean\_death\_by\_starving$	Mean of an exponential distribution for energy deaths.

Table 7.1: ABM Parameters explanation.

- To eat. If a bacterium agent has a substrate agent in the same cell as itself, it will eat a portion of substrate agent equal to its size multiplied by *size_percentage*. If not, it will look at neighbouring positions (north, south, east, west, north-east, north-west, south-east, south-west) and randomly pick one of them and eat from it. The bacterium agents that have eaten, increase its size by the quantity it has eaten, and its internal energy by one (until a maximum value of *max_energy*). The substrate agent eaten decreases its size by a percentage (*size_percentage*) of the eating bacterium agent.

- To reproduce. Bacterium agents reproduce by bipartition creating 2 individuals of the same characteristics as the parent, size half of the parent, according to a normal distribution N(*rep_size*, 1). The reproduction process follows the *deterministic cell size division approach*, that is, bacteria divide once a biomass/size threshold is reached [51]. The reproduction process starts when the bacterial size is greater than a value drawn from a normal distribution $\mathrm{N}(rep\_size, 1)$.

- To die. Bacterium agents die by two causes: running out of internal energy and age. At every step of the simulation, the bacterial age is compared with a value drawn from a normal distribution N(*max_ age*, *sd_ death_ lmax*) when the age of the bacteria is less than the maximum, or drawn from a normal distribution N(*max_ age*, *sd_ death_ gmax*) in the opposite case (an asymmetric Gaussian is defined). If the bacterial age is greater than the calculated value, the bacterium agent dies. The other cause of death is the total depletion of the bacterial internal energy of bacteria after the endogenous catabolism process (process explained in more detail at section 4.3), in which agents are unable to obtain food and they catabolise the protoplasm stored to maintain their energy. This death rate is modelled as an exponential distribution which mean is *mean_death_by_energy*.

- To update internal energy. At each step of the program, the internal energy of each bacterium agent is decremented by *d_ energy_ by_ starving* and age increases by one.

7.2.2 SSABR Description

In this subsection, we present the description of the model using the SSABR.

In this model there are two types of agents, bacterium agents and substrate agents. Bacterium agents have five states: *x coordinate*, *y coordinate*, *size*, *age*, and *internal energy*, and eleven parameters explained at table 7.1. Substrate agents have three states: *x coordinate*, *y coordinate*, and *size*; and only one parameter: *s_density*.

Bacterium Agents

$j = 1$, bacterium agent type

$\vec{x}^1_{i_1} \in \mathbb{R}^5 \; i_1 = 1, ..., n_1(t)$

$$\vec{x}^1_{i_1} = \begin{pmatrix} x \; coordinate \\ y \; coordinate \\ size \\ age \\ internal \; energy \end{pmatrix}$$

$X^1(t) = \left(\vec{x}^1_1 \; ... \; \vec{x}^1_{n_1(t)}\right); \; X^1(t) \in M_{5\times n_1(t)}(\mathbb{R})$

Partition for convenience:

$$X^1(t) = \begin{pmatrix} x^1_1(t) \\ x^1_2(t) \\ x^1_3(t) \\ x^1_4(t) \\ x^1_5(t) \end{pmatrix}; \; x^1_1(t), \; ..., \; x^1_5(t) \in M_{1\times n_1(t)}(\mathbb{R})$$

$\vec{p}^1_1 = \vec{p}^1_2 = \; ... \; = \vec{p}^1_{n_1(t)} \equiv \vec{p}^1$ All agents share the same set of parameters

$\vec{p}^1 \in \mathbb{R}^{11}$

$$\vec{p}^1 = \begin{pmatrix} max\_energy \\ i\_energy\_by\_eating \\ d\_energy\_by\_starving \\ unit\_growth \\ b\_density \\ size\_percentage \\ max\_age \\ sd\_death\_lmax \\ sd\_death\_gmax \\ mean\_death\_by\_energy \\ rep\_size \end{pmatrix}$$

Descriptions of states and parameters' meaning can be found at tables 7.2 and 7.1.

Substrate Agents

$j = 2$, Substrate Agent

$\vec{x}^2_{i_2} \in \mathbb{R}^3 \; i_2 = 1, ..., n_2(t)$

$$\vec{x}^2_{i_2} = \begin{pmatrix} x \; coordinate \\ y \; coordinate \\ size \end{pmatrix}$$

$X^2(t) = \left(\vec{x}^2_1 \; ... \; \vec{x}^2_{n_2(t)}\right); \; X^2(t) \in M_{3\times n_2(t)}(\mathbb{R})$

Agent type	State	Description
bacterium / substrate	*x coordinate*	x coordinate of the position in the environment
bacterium / substrate	*y coordinate*	y coordinate of the position in the environment
bacterium / substrate	*size*	volume occupied by the agent in the environment
bacterium	*age*	number of periods the agent has lived
bacterium	*internal energy*	number of iterations the agent can exist without eating and not dying

Table 7.2: Agent states

$$X^2(t) = \begin{pmatrix} x_1^2(t) \\ x_2^2(t) \\ x_3^2(t) \end{pmatrix};\ x_1^2(t),\ ...,\ x_3^2(t) \in M_{1\times n_2(t)}(\mathbb{R})$$
$$\vec{p}_1^{\,2} = \vec{p}_2^{\,2} = \ ...\ = \vec{p}_{n_2(t)}^{\,2} \equiv \vec{p}^{\,2};\quad \vec{p}^{\,2} \in \mathbb{R};\quad \vec{p}^{\,2} = \left(\ s\_density\ \right)$$

The environment

The environment is a two dimensional world, limited by its maximum width and height dimensions.

$\vec{\alpha} \in \mathbb{R}^2$

$\vec{\alpha} = \begin{pmatrix} width \\ height \end{pmatrix}$

Dynamic methods There are five methods in this implementation.

Step 1:

$Z^1 = X^1(t)\ \ c_1 = n_1(t)$

$Z^2 = X^2(t)\ \ c_2 = n_2(t)$

Step 2:

$$Z^1 = \begin{pmatrix} z_1^1 \\ z_2^1 \\ z_3^1 \\ z_4^1 \\ z_5^1 \end{pmatrix}\ Z^2 = \begin{pmatrix} z_1^2 \\ z_2^2 \\ z_3^2 \end{pmatrix}\ \bar{Z}^1 = \begin{pmatrix} \bar{z}_1^1 \\ \bar{z}_2^1 \\ \bar{z}_3^1 \\ \bar{z}_4^1 \\ \bar{z}_5^1 \end{pmatrix}\ \bar{Z}^2 = \begin{pmatrix} \bar{z}_1^2 \\ \bar{z}_2^2 \\ \bar{z}_3^2 \end{pmatrix}$$

$z_1^1,\ ...,\ z_5^1 \in M_{1\times c_1}(\mathbb{R});\ z_1^2,\ ...,\ z_3^2 \in M_{1\times c_2}(\mathbb{R});\ \bar{z}_1^1,\ ...,\ \bar{z}_5^1 \in M_{1\times \bar{c}_1}(\mathbb{R});$
$\bar{z}_1^2,\ ...,\ \bar{z}_3^2 \in M_{1\times \bar{c}_2}(\mathbb{R})$

Method 1: To shake. At each time step, agents in the environment will be rearranged in a random position inside the environment limits defined by $\vec{\alpha}$, changing their states $\bar{z}^1_1,\ \bar{z}^1_2,\ \bar{z}^2_1,\ \bar{z}^2_2$; thus modelling a system that operates in continuous stirred-tank reactor (CSTR) conditions, obtaining homogeneous characteristics in terms of substrate and biomass concentrations.

$$\begin{pmatrix} \bar{z}^1_1 \\ \bar{z}^1_2 \end{pmatrix} = F^1_{i_1,1}(\alpha_1,\ \alpha_2)$$
$$\begin{pmatrix} \bar{z}^2_1 \\ \bar{z}^2_2 \end{pmatrix} = F^2_{i_2,1}(\alpha_1,\ \alpha_2)$$

Method 2: To eat. The bacterium agent will eat preferably from a substrate in the same grid cell as itself ($z^1_1,\ z^1_2$); if that grid cell is empty, it will look for neighbour substrate agents ($z^2_1,\ z^2_2$) and select one randomly to eat from. It will eat a portion of substrate agent equal to its size (z^1_3) multiplied by *size_percentage* (p^1_6), decreasing the size of the substrate agent (from z^2_3 to $\bar{z}^2_3$). Bacterium agents that have eaten, increase its size (from z^1_3 to $\bar{z}^1_3$) by the quantity *unit_growth* (p^1_4), and its internal energy (from z^1_5 to $\bar{z}^1_5$) by *i_energy_by_eating* (p^1_2) until a maximum *max_energy* (p^1_1) is reached. All bacterium agents decrease its internal energy by *d_energy_by_starving* (p^1_3) due to maintenance activities.

$$\begin{pmatrix} \bar{z}^1_3 \\ \bar{z}^1_5 \\ \bar{z}^2_3 \end{pmatrix} = F^1_{i_1,2}\left(\ z^1_1,\ z^1_2,\ z^1_3,\ z^1_5,\ z^2_1,\ z^2_2,\ z^2_3,\ p^1_1,\ p^1_2,\ p^1_3,\ p^1_4,\ p^1_6\ \right)$$

Method 3: To reproduce. Bacterium agents will be divided in half creating two individuals of the same characteristics as the parent, and half the size of the parent modifying the states from z^1_3 to $\bar{z}^1_3$, according to the probability of a random variable with normal distribution of mean *rep_size* (p^1_{11}) and variance 1. As a consequence, c_1 would increase by one at every reproduction.

$$(\bar{Z}^1) = F^1_{i_1,3}\left(z^1_3,\ p^1_{11}\right)$$

Method 4: To die. It is checked whether bacterium agents will die from one of the two causes in this model: age (z^1_4), with a probability from a normal distribution of mean *max_age* (p^1_7) and different variances depending if the agent age is greater than *max_age* (*sd_death_gmax* (p^1_9)) or if the agent age is lower (*sd_death_lmax* (p^1_8)); or running out of internal energy (z^1_5), with a probability from a exponential distribution of mean *mean_death_by_energy* (p^1_{10}). If they meet the requirements for dying, they die, becoming their mass part of the food present in the reactor. Consequently, both bacterium ($\bar{Z}^1$) and substrate ($\bar{Z}^2$) sets of agents change in the number of individuals, increasing c_2 and decreasing c_1.

$$\begin{pmatrix} \bar{Z}^1 \\ \bar{Z}^2 \end{pmatrix} = F^1_{i_1,4}(z^1_4,\ z^1_5,\ p^1_7,\ p^1_8,\ p^1_9,\ p^1_{10})$$

Parameter	Value	Parameter	Value
$S0(mg/l)$	50	$\mu_{max}(day^{-1})$	1.04
$X0(mg/l)$	10	$K_s(mg/l)$	100
$Y(mg/l)$	0.55	$K_d(day^{-1})$	0.055

Table 7.3: Monod model parametrization.

Method 5: To age. At each time step, the bacterium agents age (z_4^1) is increased by one ($\bar{z}_4^1$).

$(\ \bar{z}_4^1 \ ) = F_{i_1,5}^1(z_4^1)$

Step 3:

$X^1(t+1) = \bar{Z}^1 \quad n_1(t+1) = \bar{c}_1$

$X^2(t+1) = \bar{Z}^2 \quad n_2(t+1) = \bar{c}_2$

Aggregated variables

In this model we are calculating two aggregated variables: biomass and substrate concentrations ($\vec{y}(t) \in \mathbb{R}^2$).

Biomass concentration depends on the sum of the bacterium agent sizes x_3^1 multiplied by their density p_5^1 and referred to the dimensions of the environment $\vec{\alpha}$.

$y_1(t) = g_1(x_3^1(t), \ p_5^1, \ \vec{\alpha})$

Substrate concentration depends on the sum of the substrate agent sizes x_3^2 multiplied by their density p_1^2 and referred to the dimensions of the environment $\vec{\alpha}$.

$y_2(t) = g_2(x_3^2(t), \ p_1^2, \ \vec{\alpha})$

7.2.3 Results

The model was implemented in NetLogo. To validate the behaviour of the model we have used model alignment [64], so we compared the dynamics of the ABM with the dynamics of a worldwide accepted although simple model for the process, the Monod model.

The simulations presented below show that an agent-based model is able to represent the dynamics of certain type of processes with results as good as traditional modelling paradigms, such as, in this case, the Monod model. The aim of this work is, in terms of its microscopic constituents, to understand the complex behaviour of the macroscopic system. For the Monod model parametrization, we use typical values from [112], as table 7.3 shows. The parameter values for the agent-based model were chosen using the trial and error technique; they are shown at table 7.4. They were chosen so the

Parameter	Value		
number of bacterium agents	1090	*rep_size*	7
number of substrate agents	545	*size_percentage*	0.6
initial_b_size	1	*i_energy_by_eating*	1.7
initial_s_size	10	*d_energy_by_starving*	0.7
b_density	1	*unit_growth*	0.25
s_density	1	*max_age*	50
initial_age	1	*sd_death_lmax*	20
sd_age	1	*sd_death_gmax*	10
max_energy	10	*mean_death_by_starving*	2
sd_energy	2		

Table 7.4: ABM parametrization.

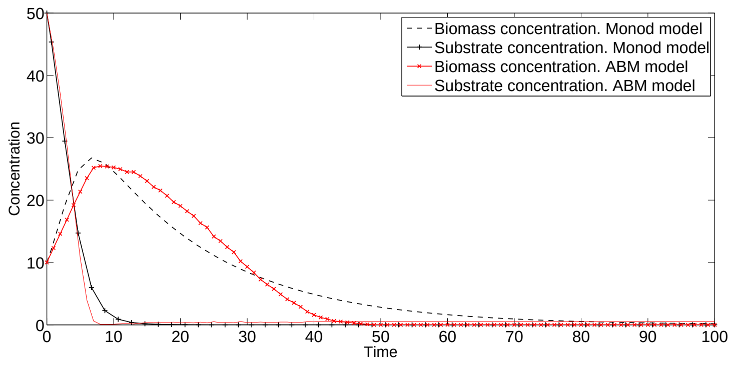


Figure 7.2: Temporal evolution of the concentrations of substrate and biomass in ABM and Monod models.

model response behaves in a similar way as Monod model response. The responses of both models are shown at figure 7.2. Moreover, figure 7.2 reproduces the different phases of the activated sludge process that were explained before, except for the lag phase, because the microorganisms in this model are initially accommodated to the environment.

Figure 7.3 and figure 7.4 show the evolution over time of bacteria reproduction process and bacteria deaths, respectively. The reproduction process takes place from the twenty-three simulation step to the forty-seven, which can be considered the 2-3 phases (section 4.3), as figure 7.3 shows. In figure 7.4 the two causes of death can be observed.

To test the model, a set of experiments were done, comparing the experimental results with the expected behaviour and respect to the nominal case (figure 7.2).

For example, if *size_percentage* parameter (percentage that applied to

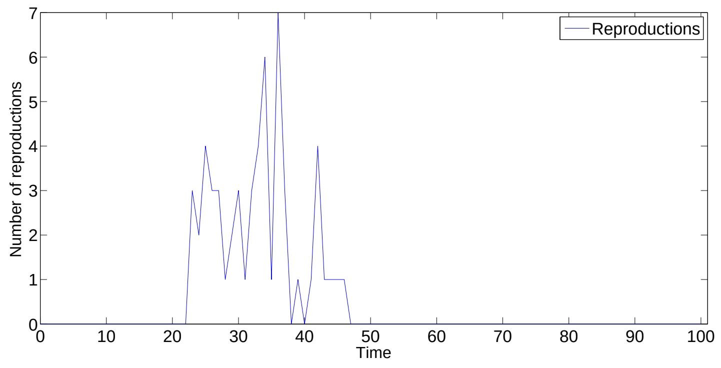


Figure 7.3: Temporal evolution of bacteria reproduction process in the ABM.

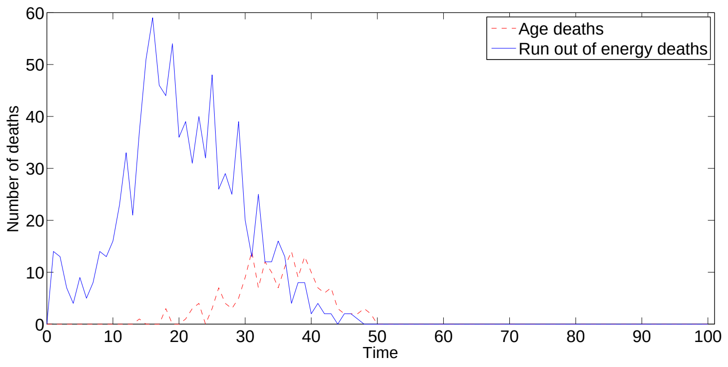


Figure 7.4: Temporal evolution of bacteria deaths in the ABM.

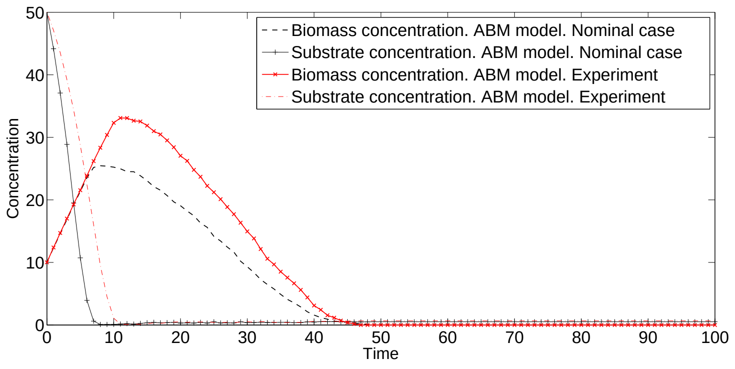


Figure 7.5: Temporal evolution of the concentrations if *size_percentage* is decreased by 50%.

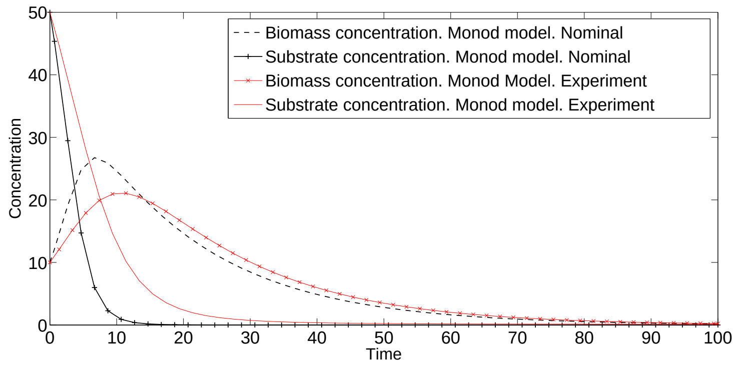


Figure 7.6: Comparison: Monod model base case dynamics Vs. experiment (μ_{max}=1, K_s=200).

the size of a bacterium agent indicates the amount that will decrease the size of a substrate agent to be eaten by that bacterium agent) is decreased by 50%, the evolution of substrate and biomass concentrations is as shown in figure 7.5. The bacterium agents eat a smaller portion of substrate each time they eat, thus the food present in the environment (substrate) last longer than in the nominal case. For this reason, the biomass concentration reaches a higher maximum value than in the first case, being in the environment more agents (if agents reach the reproduction size, they can reproduce, so the more food, the more bacterium agents). When the substrate ends, stationary phase starts, followed by the endogenous phase. The evolution of these two phases is slower than in the nominal case because there are more bacterium agents.

If we want to change the growth rate of bacteria, with the Monod model we could manipulate the parameters μ_{max} and K_s. These are empirical coefficients of the Monod equation. They will differ between species and based on the ambient environmental conditions. As they are related, we must change both together. In figure 7.6 we performed a change from the nominal values $\mu_{max} = 1.04$ and $K_s = 100$ to $\mu_{max} = 1$ and $K_s = 200$. The new values cause a decrement in the bacteria growth rate, so the substrate last longer than in the nominal case (figure 7.6).

With our ABM, we can influence the bacteria growth rate manipulating the parameters *unit_growth* and *rep_size*.

A decrement in *unit_growth* (amount by which the size of a bacterium agent is increased when eating) causes that the bacterium agents grow slower than in the nominal case, so they metabolize smaller portions of substrate and the total depletion of substrate occurs later than in the nominal case, as figure 7.7 shows.

A smaller value in *rep_size* (the bacterium agents start the reproduction

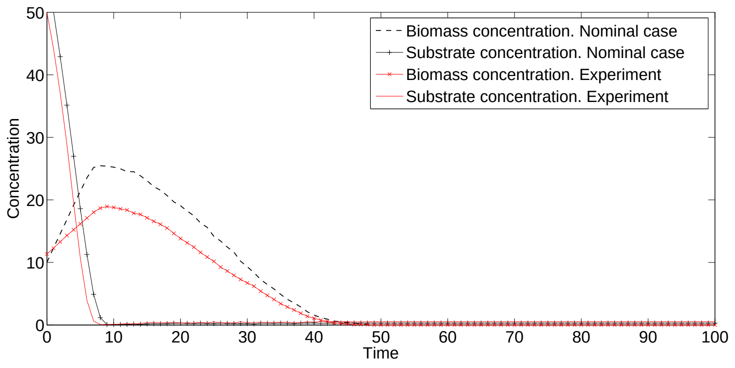


Figure 7.7: Comparison: ABM model base case dynamics Vs. experiment ($unit\_growth$=0.1).

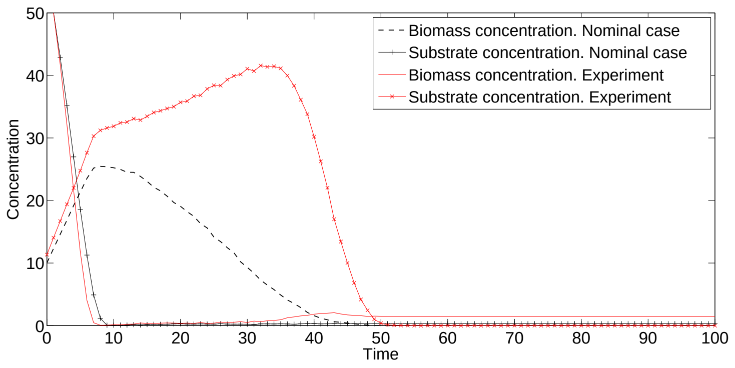


Figure 7.8: Comparison: ABM model base case dynamics Vs. experiment (rep_size=4.5).

process earlier) causes that the population increases in number significantly, reaching high values of biomass concentration (figure 7.8).

If we want to change the decay rate of bacteria, with the Monod model we can conduct a change in K_d parameter, as for example in figure 7.9 where K_d is increased by 82%; and manipulating the yield coefficient Y (conversion coefficient between cell growth rate dX/dt and substrate utilization rate dS/dt), as figure 7.10 shows.

In our ABM we can expect similar results manipulating the parameters max_age, $i\_energy\_by\_eating$, $d\_energy\_by\_starving$ and max_energy.

If a 20% of decreasing is performed in max_age, so bacterium agents may die younger than in the nominal case, the results are shown in figure 7.11 and figure 7.12. As it can be observed in figure 7.12, bacterium agents start dying earlier than in the nominal case, as expected. This causes the 'food' (substrate) to last longer and the logarithmic phase ends later. Moreover,

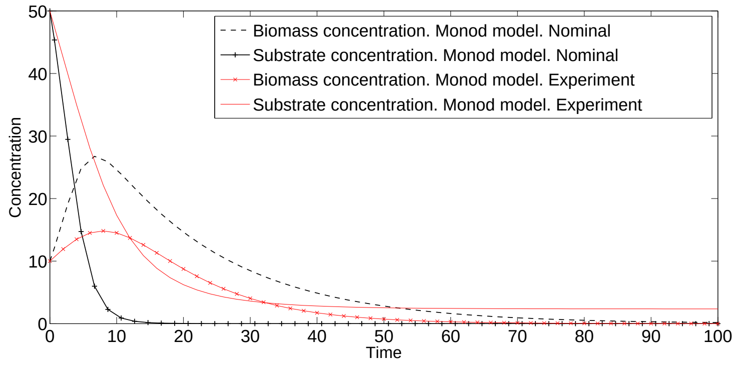


Figure 7.9: Comparison Monod model base case dynamics Vs. experiment (K_d=0.1).

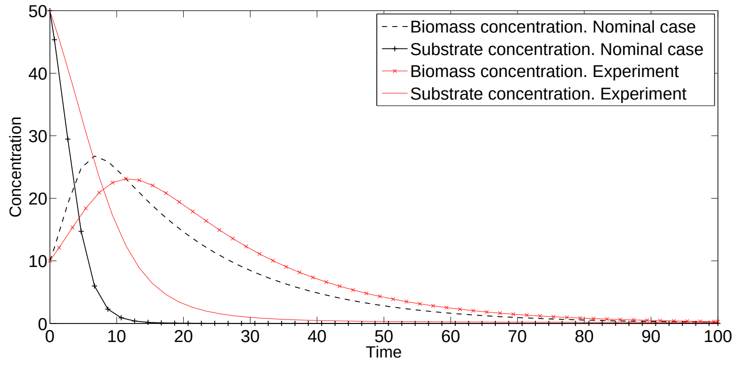


Figure 7.10: Comparison: Monod model base case dynamics Vs. experiment (Y=0.65).

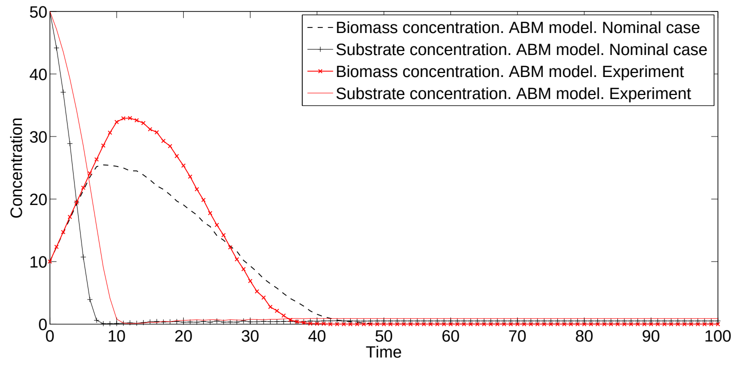


Figure 7.11: Temporal evolution of the concentrations if max_age is decreased by 20%

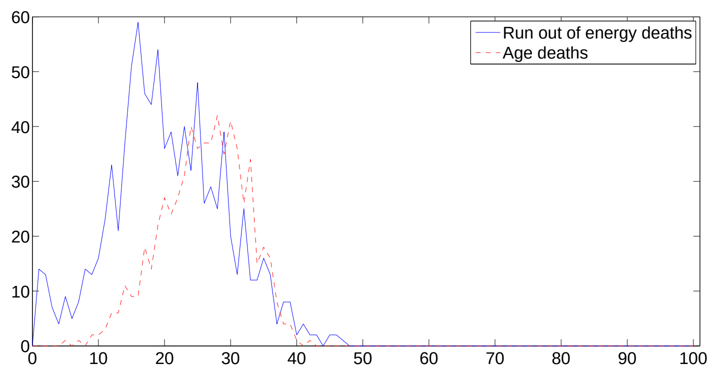


Figure 7.12: Temporal evolution of bacteria deaths if *max_age* is decreased by 20%.

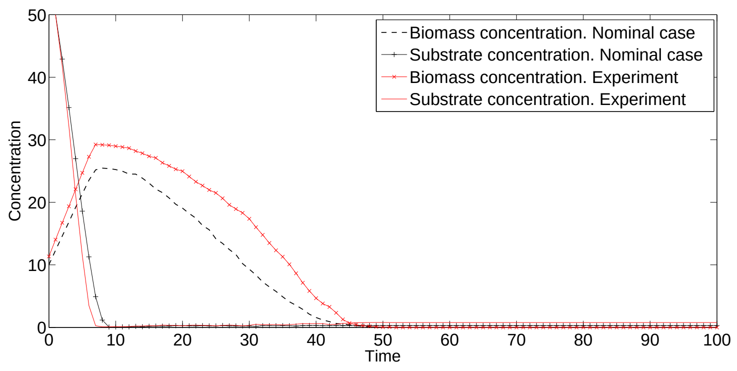


Figure 7.13: Comparison: ABM model base case dynamics Vs. experiment (*i_energy_by_eating*=2.5).

the substrate concentration reaches zero (all substrate agents were eaten) earlier than in first case, as figure 7.11 shows.

The amount of internal energy a bacterium agent gains every time it eats (*i_energy_by_eating*) also influences this process. The bigger the increase in internal energy, the stronger a bacterium agents is (so it would die older). The amount of internal energy a bacterium agent lose every time step (*d_energy_by_starving*) has an opposite influence. Both parameter influences are shown in figures 7.13 and 7.14 respectively.

The internal energy of a bacterium agent cannot be increased indefinitely, but to a maximum (*max_energy*). This parameter is related with the two previous ones and also influences the viability of bacteria, as figure 7.15 shows, where an increase is performed in *max_energy* so the bacterium agents start dying later than in the nominal case.

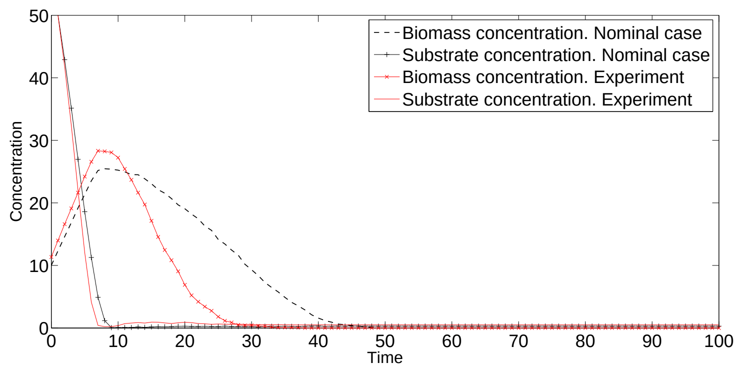


Figure 7.14: Comparison: ABM model base case dynamics Vs. experiment ($d\_energy\_by\_starving$=1.4).

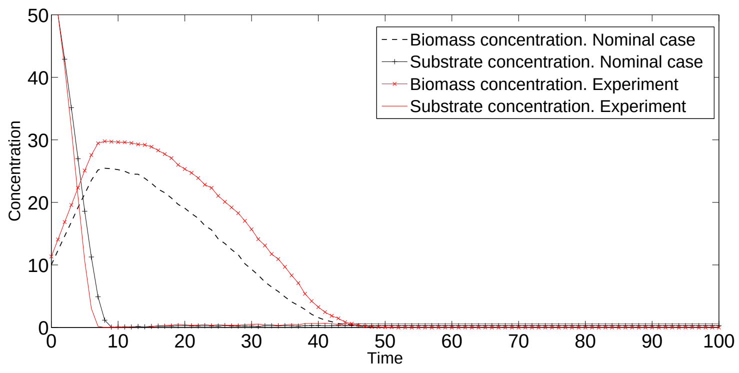


Figure 7.15: Comparison: ABM model base case dynamics Vs. experiment (max_energy=12).

7.2.4 Conclusions

This first version of the model allowed us to test the behaviour of the assumptions made in the conceptualization process. This work was presented at the 19th Mediterranean Conference on Control Automation (MED) that was held in Corfu (Greece) in 2011 [100]. This work showed that ABM allows the researcher to develop a bottom-up approximation to the study of complex problems such as the phenomena occurring in a wastewater treatment plant, and in particular, secondary treatment by activated sludge. An agent-based model for the activated sludge process in a batch reactor has been proposed, which provides a better understanding of the phenomena that occurs in the system while varying some of its characteristics.

7.3 Activated Sludge Batch Reactor Model. MATLAB version

As in a general purpose programming language, the representation of agents in MATLAB can be carried out using different alternatives:

1. *Layered representation* (figure 7.16a), where the world is modelled with different matrix layers, each layer representing an agent property and the position of each element of the matrix determines the location of the agent in the world. It is inspired in the automata cellular approach. This alternative has an important limitation; the agents positions in the world are restricted to the matrix elements coordinates.

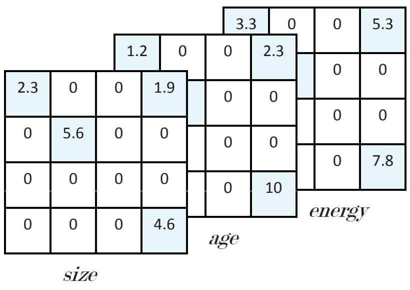
2. To solve the disadvantage of the previous Layered representation, the agents can be modelled having a property representing their position in the world, relative to a Cartesian axe. Two approaches can be used. On the one hand, *data structures*, where the properties of the agents can be accessed with the dot notation, e.g., *agent(number).property* (figure 7.16b).

3. On the other hand it is the *matrix representation.* The agent properties can be stored in a matrix *agent(number of agent, number of property)* (figure 7.16c).

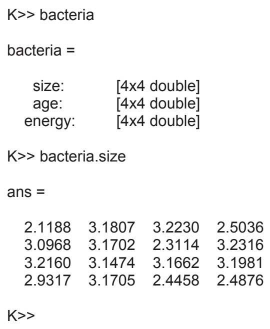
4. Object Oriented Programming (OOP) (figure 7.16d). It is a programming paradigm that is based on *objects*, data structures consisting of data fields and methods together with their interactions. A generic agent is represented as an object, and the other agents are created inheriting their characteristics from that agent, and adding their owns. This can be done due to the inheritance property of OOP (inheritance is a way to reuse code of existing objects, establishing a subtype from an existing object).

In this thesis we have compared the four implementation alternatives versus the NetLogo implementation from section 7.2, looking for the one with less computational time. This was motivated by the future possible application in automatic control, where the models are used in real time for computing the next control action, so computational time is a limitation factor in a practical implementation. In table 7.5 we present the average simulation time of ten executions for each experiment and for our implementations. The model implemented with the four alternatives is the same; it is a variation from the one in section 7.2. It is described in the following subsection 7.3.1.

The alternative with less computational time (in the majority of cases) and consequently selected was the matrix representation (figure 7.16c), which takes advantage of the computing power of vectorial calculus of MATLAB.

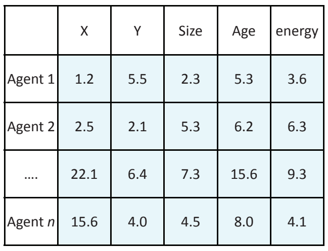


(a) Layered representation. Alternative 1



```
K>> bacteria

bacteria =

      size:    [4x4 double]
       age:    [4x4 double]
    energy:    [4x4 double]

K>> bacteria.size

ans =

    2.1188   3.1807   3.2230   2.5036
    3.0968   3.1702   2.3114   3.2316
    3.2160   3.1474   3.1662   3.1981
    2.9317   3.1705   2.4458   2.4876

K>>
```

(b) Data structure representation. Alternative 2



	X	Y	Size	Age	energy
Agent 1	1.2	5.5	2.3	5.3	3.6
Agent 2	2.5	2.1	5.3	6.2	6.3
....	22.1	6.4	7.3	15.6	9.3
Agent *n*	15.6	4.0	4.5	8.0	4.1

(c) Matrix representation. Alternative 3

```
classdef agent
    properties
        position
        size
        color
        shape
        label
    end

    methods
        function [obj, child] = hatch(obj)
            obj.size = obj.size / 2;
            child = obj;
        end

        function obj = move (obj, position)
            obj.position = position;
        end
    end
end
```

(d) OOP representation. Alternative 4

Figure 7.16: ABM in MATLAB, alternatives: (a) Layered representation, (b) Data structure, (c) Matrix representation, (d) OOP representation.

The OOP alternative is a suitable natural representation of agents. In fact, it is widely adopted as the most common paradigm for ABM frameworks. Our implementation under this paradigm was presented in the conference *9th International Symposium on Distributed Computing and Artificial Intelligence*, in Salamanca (Spain) the 28-30th March of 2012. It is now published as a book chapter [102]. We have also developed a graphic interface for the configuration of the simulations (figure 7.17).

7.3.1 Description

This model is an improvement from the previous one in section 7.2. In this section we only explain the differences from the previous one.

In *Entities, state variables, and scales*: Each bacterium is represented with five variables: location (**two variables** for the two-dimensional coordinates in a Euclidean space limited by the parameters *width* and *height*),

Experiment	Implementations				
	N. [a]	L.R. [b]	D.S. [c]	M.R. [d]	O.O.P. [e]
220-150-30[f]	1.504	2.339	1.532	0.163	1.723
220-150-60	2.213	2.005	1.509	0.449	2.319
440-300-30	1.702	4.309	2.838	0.523	3.581
2220-1500-30	2.252	7.461	4.532	2.514	5.993
7333-5000-30	6.129	22.502	19.474	15.963	27.759

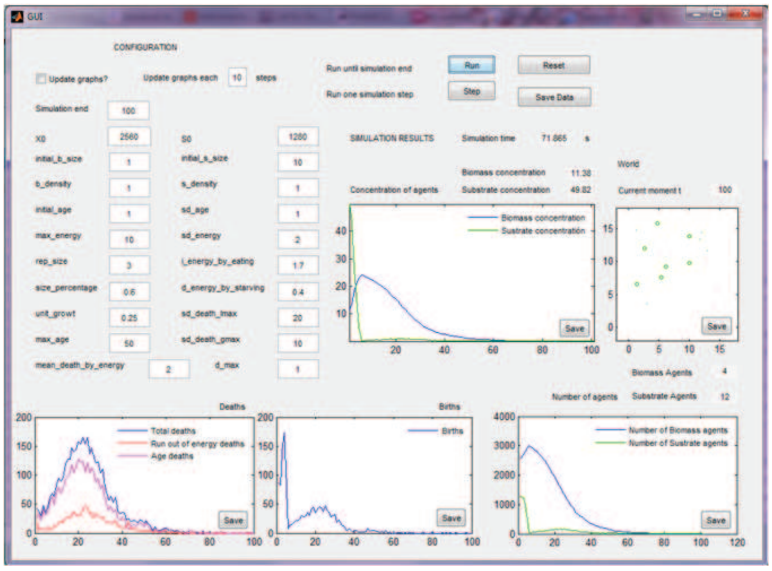
Table 7.5: Simulation time (seconds) comparison of implementations.

[a] N: NetLogo
[b] L.R.: Layered representation
[c] D.S.: Data structure
[d] M.R.: Matrix representation
[e] O.P.P.: Object Oriented Programming
[f] The codification X-X-X of the experiments means: number of bacterium agents - number of substrate agents - width/height of the square environment.

size, age, and internal energy. The substrate units are represented by agents with the state variables location (**also two variables**) and size.

In *Details*: The initialization process is conducted by the user who assigns values to the constants of the model. Then, the model creates the agents and places them in the world, and assigns initial values for agent variable states. Initially, agents are placed randomly on the environment as a uniform distribution. Bacterium agents are created with initial values of energy as a normal distribution $N(max\_energy, sd\_energy)$, initial age values as a normal distribution $N(initial\_age, sd\_age)$, and constant size ($initial\_b\_size$). Substrate agents are created with constant size ($initial\_s\_size$). The inputs that have to be set up by the user in the interface are the ones in table 7.1 and **the new one** eat_radius **(a bacterium agent can only eat in its surrounding)**.

In *Submodels*: *To eat.* **A bacterium agent can only eat in its surroundings defined by a circle of radius** eat_radius. It will eat a portion of substrate agent equal to its size multiplied by size_percentage. The bacterium agents that have eaten, increase its size by the quantity it has eaten, and its internal energy by one (until a maximum value of max_energy). The substrate agent eaten decreases its size by a percentage ($size\_percentage$) of the eating bacterium agent.



Update graphs? ☐ Update graphs each 10 steps

Run until simulation end

Run one simulation step

Simulation end	100		
X0	2560	S0	1280
initial_b_size	1	initial_s_size	10
b_density	1	s_density	1
initial_age	1	sd_age	1
max_energy	10	sd_energy	2
rep_size	3	i_energy_by_eating	1.7
size_percentage	0.6	d_energy_by_starving	0.4
unit_growt	0.25	sd_death_lmax	20
max_age	50	sd_death_gmax	10
mean_death_by_energy	2	d_max	1

SIMULATION RESULTS Simulation time 71.865 s

Concentration of agents Biomass concentration 11.38 Substrate concentration 49.82

Current moment t 100

Number of agents Biomass Agents 4 Substrate Agents 12

Figure 7.17: Graphic User Interface in MATLAB.

7.3.2 Results

To analyse the behaviour of the model we carried out the same experiments as with the NetLogo implementation, comparing the dynamics of the ABM with the dynamics of the Monod model. In the following, we analyse the effect on the emergent macroscopic behaviour of the model when changing some of the parameters of the model. Nominal behaviour, for comparison, is as figure 7.18 shows.

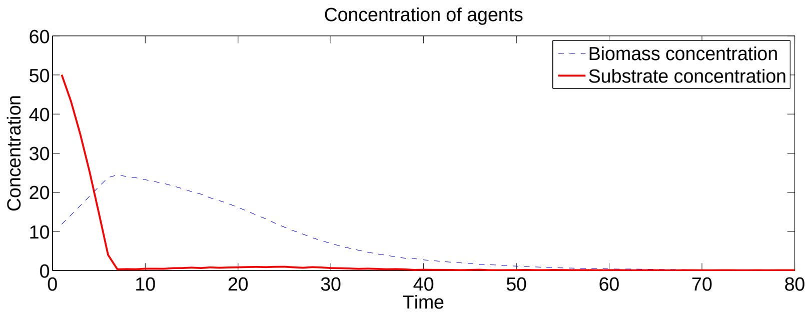
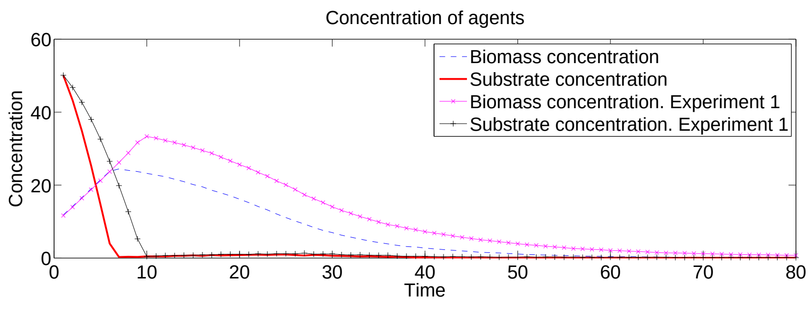
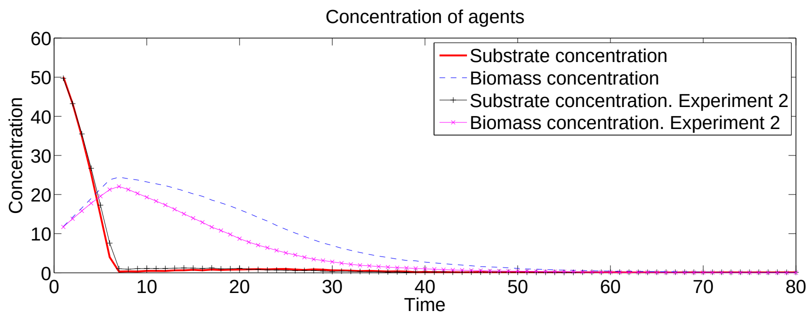


Figure 7.18: Evolution of concentration of agents over time. Nominal case.

We have done the same experiments as with the NetLogo model in section 7.2 inferring the same results, as expected. We only show a couple of examples, figures 7.19a and 7.19b show the same two experiments as in the previous NetLogo version (figures 7.5 and 7.11).



(a) *size_ percentage* parameter effect.



(b) *max_ age* parameter effect.

Figure 7.19: Concentration of agents in experiments.

7.4 Activated Sludge Reactor Model. Improved MATLAB version

This model is based on the previous one, but includes some changes and improvements thanks to the work with Marta Ginovart from the MOSIMBIO (Discrete Modelling and Simulation of Biological Systems) group at the *Universitat Politècnica de Catalunya* during my stay in Barcelona the months of September to November of 2012. The MOSIMBIO group studies different microbial systems (such as bacteria, yeast, protozoan parasites or multispecies ecosystems) and processes of industrial or environmental interest. Using the ABM (or Ibm) methodology, they have developed INDISIM (INDividual DIScrete SIMulations) [38], a model to simulate the growth and behaviour of bacterial populations. INDISIM has been used in many works [37], [38], [39] and it has several adaptations [30], [35]. This simulator has inspired the improvements made on our model, and so have done the advices from Marta Ginovart.

Besides the batch operation protocol, we added another three protocol possibilities to the simulator: continuous regime, fed-batch and semi-continuous. In the continuous culture regime, there is a continuous supply of substrate and bacteria to the reactor; the fed-batch involve a discontinued substrate feeding and the products remain in the bioreactor until the end of the run; and the forth one is the one we have called *semi-continuous* and it is equal to fed-batch, but in the feeding, bacteria are also introduced.

As this model is an evolution from the previous one, in the following we only explain the improvements.

In *Entities, state variables, and scales*: The basic entities of the model are bacteria and substrate units. Each bacterium is represented with five state variables: two coordinates of location in a two-dimensional Euclidean space (limited by *width* and *height* parameters), mass, **viability (whenever the environmental conditions become unfavourable, the bacterium may lose its viability and stop being metabolically active)**, and internal energy. The substrate units are represented by agents with the state variables: two coordinates of location, and size.

In *Process overview and scheduling*: At each time step of the simulation the following actions are performed: reactor stirring, **bacterial uptake and maintenance**, bacterial reproduction, **to update bacterial viability**, **and feeding and removal of medium** (depending on the operation protocol selected). The process of updating the state variables is asynchronous.

The global scheduling of the simulation follows different steps: initialization of variables, **operation protocol selection**, execution of the simulation and generating and displaying the results.

In *Design concepts*:

Emergence. The dynamics of the system are a result of the interaction between bacteria and substrate. The structure of the bacterial population (parametrized in the initialization process), the spatial environment and the **operation protocol** determine the dynamics of the system.

Adaptation. Bacteria reduce their own biomass when there is a lack of nutrient, until a limit. If this limit is surpassed, bacteria die. In this model, the inhibitory effect produced by the end product excreted by the bacteria is not contemplated.

Stochasticity. Stochasticity is used to provide the bacterial population with heterogeneity characteristics, initializing their characteristics around mean values, using a normal distribution (the most commonly encountered distribution in nature). It is also used when applying the rules that govern the model, modelling the stochasticity that is present in nature. Variability is an intrinsic quality of the modelled system. **To reduce the number of parameters in our model, we have fixed the stochasticity of all the processes to the same proportion**, that is, a twenty percent of their mean. This wilful value should be revised if the dynamics of the model are not adequate, and should also be calibrated to a real system when using the model in practice. From now on, all normal distributions used have a standard deviation with value twenty percent of their mean.

In *Details*:

Initialization. The initialization process is conducted by the user who assigns values to the constants of the model. Then, the model creates the agents and places them in the world, and it assigns initial values of agent variable states. Initially, agents are placed randomly on the environment as a uniform distribution.

Bacterium agents are created with initial values of mass as a normal distribution centred at *rep_size* $\mathrm{N}(rep\_size, 20\% rep\_size)$, **viability and internal energy properties are initialized to zero**. Substrate agents are created with initial values of mass as a normal distribution centred at *initial_s_size* $\mathrm{N}(initial\_s\_size, 20\% initial\_s\_size)$. **The initial number of bacterium agents and substrate agents can be initialized by two ways, just setting *number of bacterium agents* and *number of substrate agents* parameters, or setting the initial concentrations of biomass ($X0i$) and substrate ($S0i$) inside the reactor so that the number of agents is deduced.**

The inputs that have to be set up by the user in the interface are: *number of bacterium agents*, *number of substrate agents*, $X0i$, $S0i$, *initial_s _size*, *b_density*, *s_density*, *rep_size*, *availability*, *uptake*, *viability*, *sd_ viability*, *yield*, *maintenance*, *flow*, $X0$, $S0$, *period*, fed_time, *eat_radius*, *width* and *height*. The meaning of the new parameters is explained in table 7.6.

Submodels. :

- Reactor stirring. At each step of the program, agents in the world are

Parameter	Value
number of bacterium agents	Number of bacterium agents inside the reactor at time step zero.
number of substrate agents	Number of substrate agents inside the reactor at time step zero.
$X0i$	Initial concentrations of biomass inside the reactor at time step zero.
$S0i$	Initial concentrations of substrate inside the reactor at time step zero.
availability	Percentage modelling the entry of nutrient particles into the bacterial cell.
uptake	Mean value of maximum nutrient a bacterium can uptake.
viability	Number of periods a bacterium could not satisfy the maintenance requirements.
sd_ viability	Standard Deviation for modelling stochasticity in bacterial dead.
yield	Yield of the process of covering maintenance requirements and increasing the biomass.
maintenance	Percentage of bacterial mass used to calculate the maintenance requirements.
flow	Flow of the influent in continuous, fed-batch and semi-continuous operating protocols.
$X0$	Biomass concentration on the influent.
$S0$	Substrate concentration on the influent.
period	Period of a square wave influent in fed-batch and semi-continuous operating protocols.
fed_time	Duration of the feeding (maximum amplitude phase of a square wave influent) in fed-batch and semi-continuous operating protocols.

Table 7.6: Improved MATLAB ABM Parameters explanation.

rearranged in a random uniform position **in a circular area centred at its previous position and with a radius** 30%*width* (30% of the environment width), thus modelling a system that operates in continuous stirred-tank reactor (CSTR conditions, obtaining homogeneous characteristics in terms of substrate and biomass concentrations.

- Bacterial uptake and maintenance. A bacterium agent can only eat in its surroundings defined by a circle of radius *eat_radius*. **The quantity of "food" a bacterium can reach is also limited by the coefficient** *availability* which takes into account the probabilistic regarding the entry of nutrient particles into the bacterial cell through the cellular membrane. **The quantity a bacterium can uptake is proportional to its cellular surface and to a coefficient of uptake according to a normal distribution N(**$uptake, 20\%uptake$**)** [105]. **This nutrient quantity is used to cover the maintenance requirements** (applying *yield* coefficient), these maintenance requirements are proportional (*maintenance* parameter) to the bacterium agent mass. **If the nutrient absorbed is not enough to cover the maintenance requirements, it is checked the possibility of bacterial lysis** (the bacterium degrades its own biomass until it achieves a minimum size). **Once the viability of the bacterium agent is achieved, if there is any remaining energy, the bacterium can increase its own biomass** (depending on the *yield* coefficient).

- Bacterial reproduction. Bacterium agents reproduce by bipartition creating 2 individuals of the same characteristics as the parent, mass half of the parent). The reproduction process follows the *deterministic cell size division approach*, that is, the bacteria divide once a biomass/size threshold is reached [51]. The reproduction process starts when the bacterial size is greater than a value drawn from a normal distribution N($rep\_size, 20\%rep\_size$).

- To update bacterial viability. **If a bacterium can not satisfy the maintenance requirements, it increases its viability property. If this value is greater than a value drawn from a normal distribution N(***viability*, 20%*sd _viability***), the bacterium dies**.

For the continuous, fed-batch and semi-continuous operating protocols, incoming agents are randomly placed on the environment, and outcoming agents are randomly selected.

The quantity of bacterium and substrate agents on the input is parametrized by the influent flow ($flow$) and the biomass and substrate concentrations on the influent ($X0$ and $S0$). In the fed-batch, $X0$=0, that is, there is only feeding of substrate. In the continuous operating protocol, the flow is equal for the influent and the effluent.

The discontinuous feeding in both fed-batch and semi-continuous is modelled with a square wave, where *period* is the period of the wave, and fed_time is the duration of the feeding. The feeding starts in time step zero.

7.4.1 Results

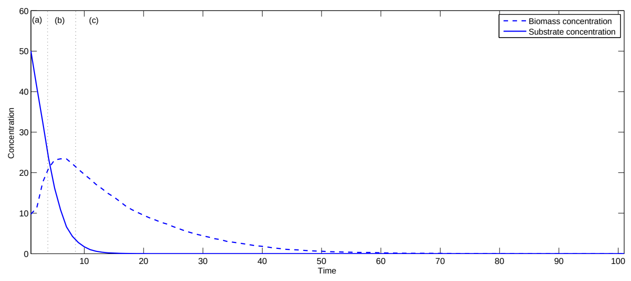
The model was implemented in MATLAB. A set of experiments were conducted to test the model.

7.4.1.1 Testing the model behaviour

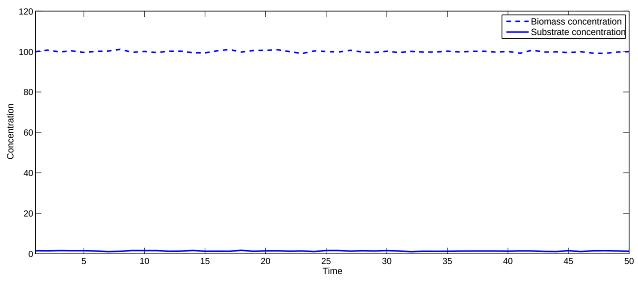
Firstly, as we did with the other previous models, we compared the dynamics of the model with the Monod model. The parametrization of the model was not an issue, since we did not want, at this stage, to tune them to mimic any reference model (which is done in the next section); just only to analyse the qualitative behaviour. The behaviour of the model was compared with the behaviour of the Monod model as in the previous versions, identifying the different phases of the activated sludge process in the case of the batch operating protocol (figure 7.20a) and a regular response for the rest of the cases (figures 7.20b, 7.20c, 7.20d).

As figure 7.20c shows, the high renewal frequency of substrate in the fed-batch operating protocol causes the bacteria in the reactor to grow and reproduce until a stationary oscillating state is reached.

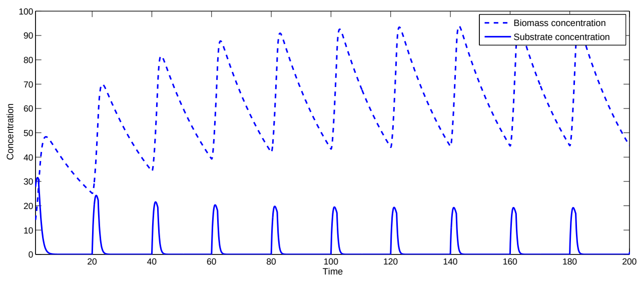
The renewal frequency in the example in figure 7.20d is also high, not reaching the biomass concentration a zero value (all bacteria die).



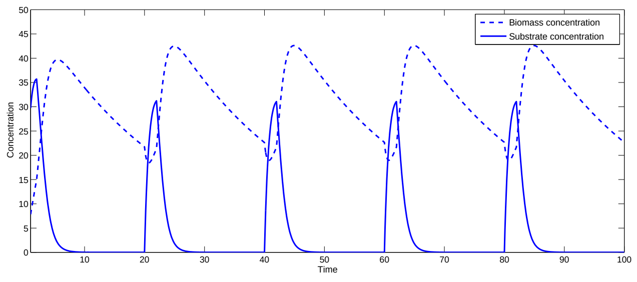
(a) ABM in batch operating protocol.



(b) ABM in continuous operating protocol.



(c) ABM in fed-batch operating protocol.



(d) ABM in semi-continuous operating protocol.

Figure 7.20: Dynamic behaviours of the four operating protocols: (a) batch, (b) continuous, (c) fed-batch, (d) semi-continuous.

7.4.1.2 Model parameters tuning for the Batch Operation Protocol

In this section we deal with ABM parameter tuning. We adjusted the model parameters for the Batch operating protocol to mimic the Monod model with a parametrization obtained from the calibration with a real system. The Monod model parameters values chosen were the ones on table 7.7 obtained from [78] (original source in persian [119]).

Parameter	Value	Parameter	Value
$S0(mg/l)$	50	$\mu_{max}(day^{-1})$	9.39
$X0(mg/l)$	10	$K_s(mg/l)$	169.3
$Y(mg/l)$	0.882	$K_d(day^{-1})$	0.107

Table 7.7: Monod model parametrization from [78].

The majority of the parameters were adjusted with reasonable values (since we are modelling real individuals), with the help of Marta Ginovart and their simulator [40]. The rest of the parameters (less than 6) were calibrated using the trial and error technique, starting with values similar to the ones Marta Ginovart used in [40].

The parameter values estimated for our ABM are shown in table 7.8. Once the model is tuned for the batch operating protocol, we have the bacterial population parameters tuned, and we can work with the other operating protocols since the intrinsic characteristics of the bacterial population (viability, reproduction size, etc) do not change with the operating protocol.

Parameter	Value	Parameter	Value
number_of_bacterium_agents	54	*maintenance*	1.95
number_of_substrate_agents	45	*viability*	500
b_density	100	*sd_viability*	0.55
s_density	100	*initial_s_size*	10
eat_radius	4.24	*width*	30
rep_size	2	*height*	30
uptake	1.95	$X0$	0
availability	0.5	$S0$	0
yield	1.95	*flow*	0
period	0		

Table 7.8: ABM model parametrization.

As we can notice in figure 7.21, the ABM behaviour perfectly fits the Monod model behaviour.

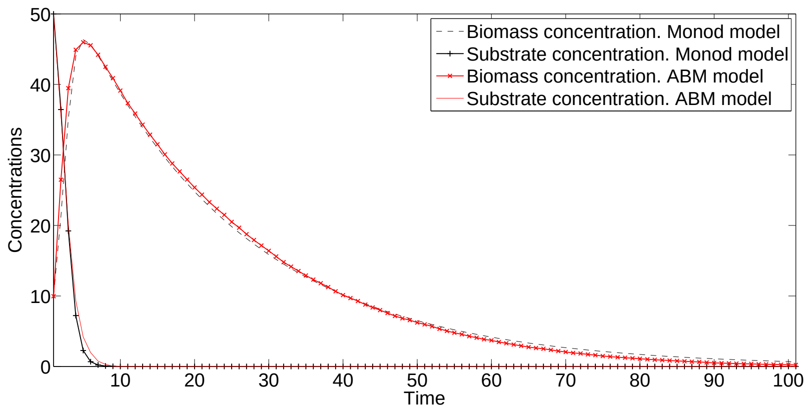


Figure 7.21: Comparison of ABM and Monod model behaviours.

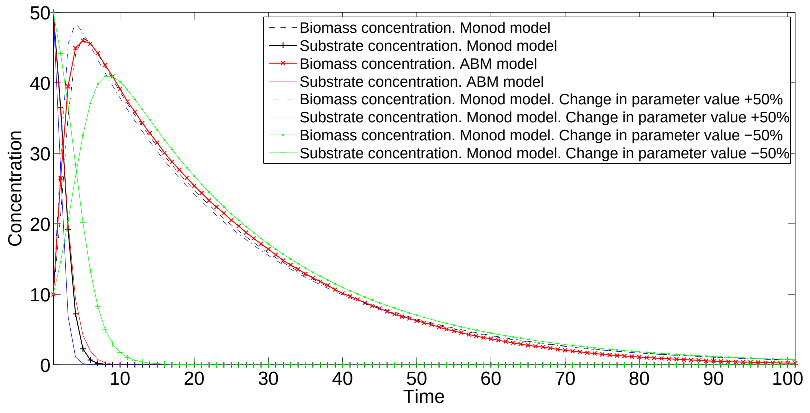
7.4.1.3 Parameters sensitivity experiments

We have studied the sensitivity of the ABM parameters and comparing it with the Monod model. A change in a parameter was done, maintaining the rest fixed. In the following we can examine some results.

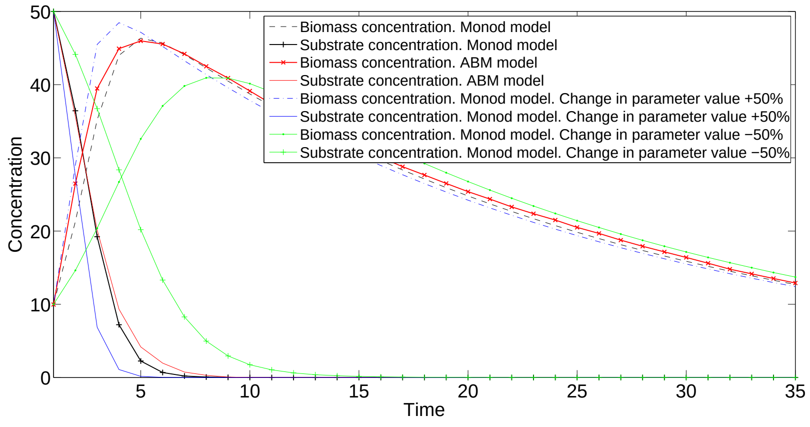
Figure 7.22 shows the behaviour of the Monod model if we manipulate the growth rate of bacteria with the parameter μ_{max} (increased by 50% (in green) or decremented by 50% (in blue)).

We can have a similar behaviour with the ABM modifying the parameters *rep_size*, *uptake*, *availability* and *yield*.

The effects of changing *rep_size* (size from which the bacteria are able to reproduce) can be observed in figure 7.23. If this value is small, the bacterium agents start reproducing with a smaller size.

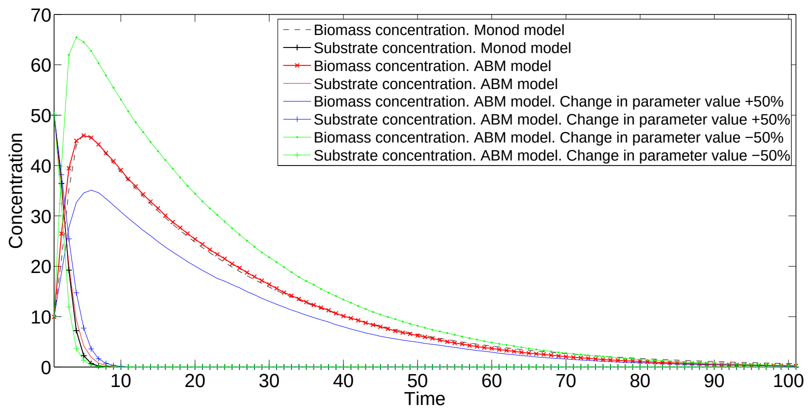


(a) Simulation time from 0 to 100

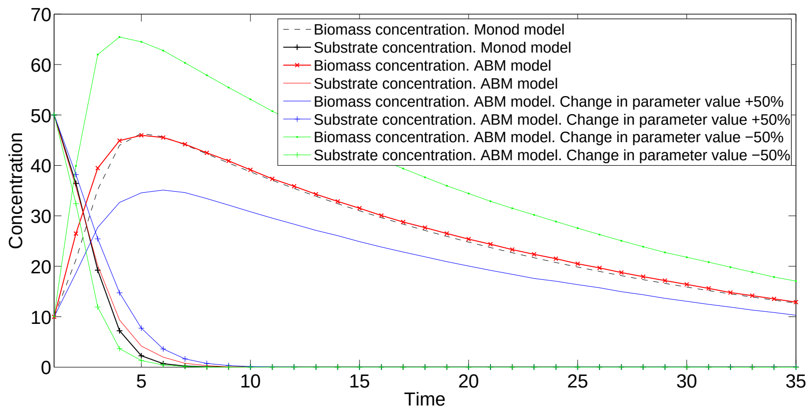


(b) Zoom to simulation time from 0 to 35

Figure 7.22: Concentration of biomass and substrate. μ_{max} parameter effect.

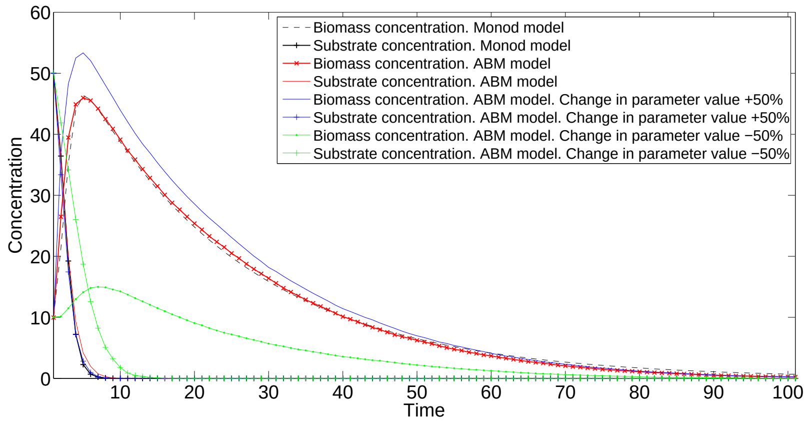


(a) Simulation time from 0 to 100

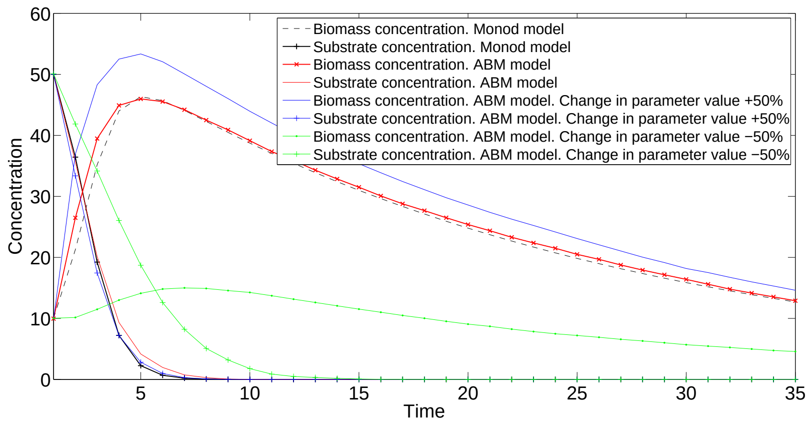


(b) Zoom to simulation time from 0 to 35

Figure 7.23: Concentration of agents. *rep_size* parameter effect.



(a) Simulation time from 0 to 100

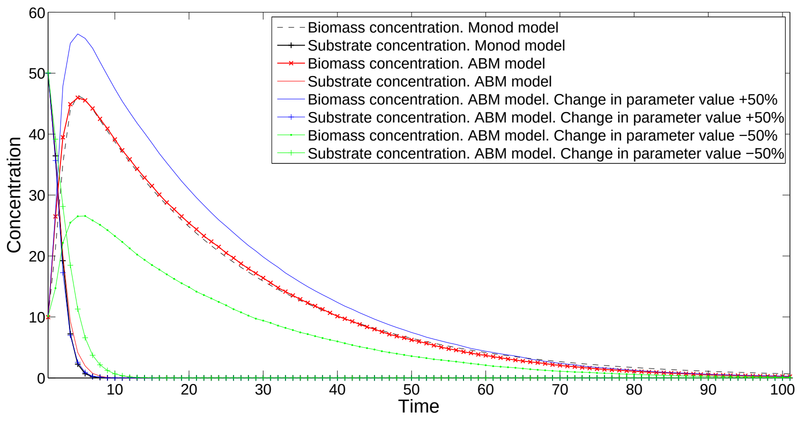


(b) Zoom to simulation time from 0 to 35

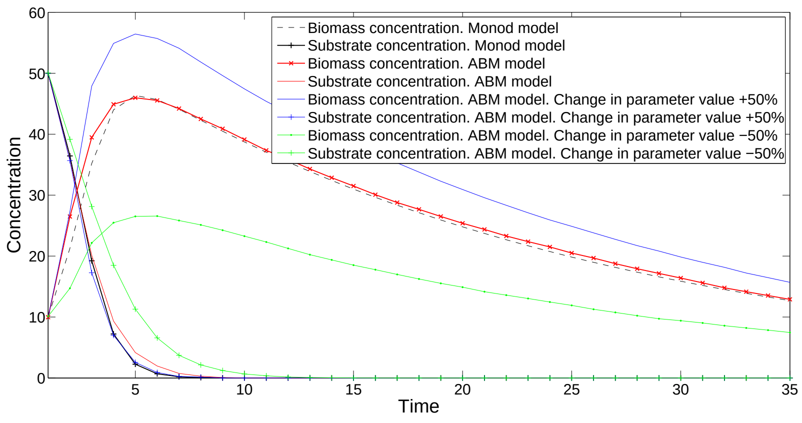
Figure 7.24: Concentration of agents. *uptake* parameter effect in the ABM.

The *uptake* parameter regulates the uptake process. A higher value allows the bacteria to metabolize more substrate, as figure 7.24 shows.

The *availability* parameter limits the quantity of nutrient a bacterium can uptake, taking into account the probabilistic regarding the entry of nutrient particles into the bacterial cell through the cellular membrane. The higher the *availability* parameter value, the more available nutrient for a bacterial cell. This effect is shown in figure 7.25.

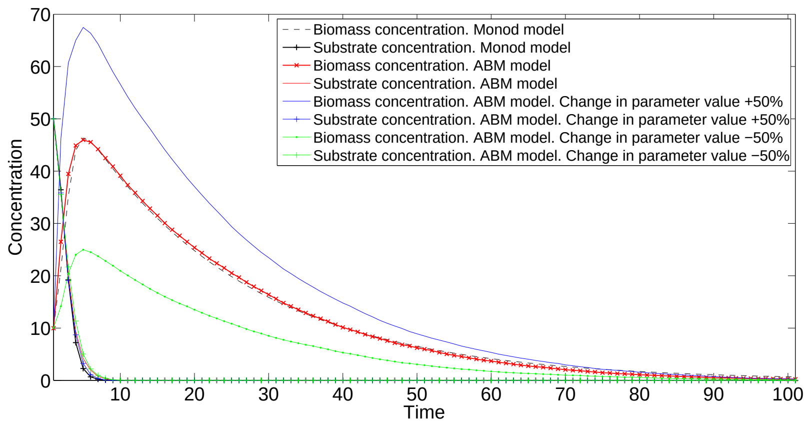


(a) Simulation time from 0 to 100

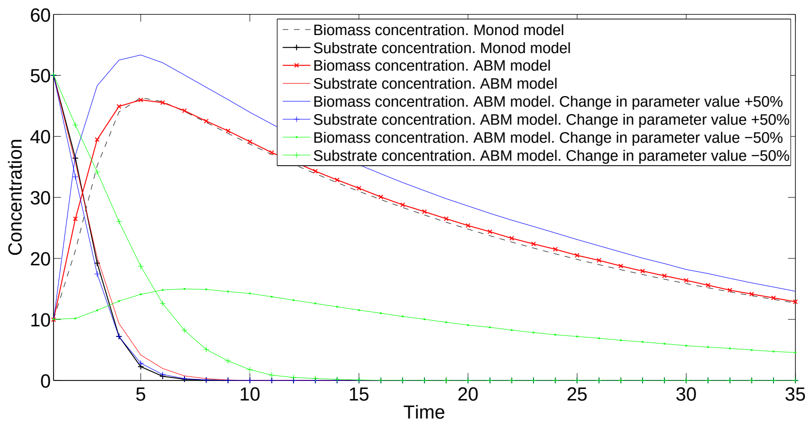


(b) Zoom to simulation time from 0 to 35

Figure 7.25: Concentration of agents. *availability* parameter effect in the ABM.



(a) Simulation time from 0 to 100



(b) Zoom to simulation time from 0 to 35

Figure 7.26: Concentration of agents. *yield* parameter effect from the ABM.

The *yield* parameter is a performance parameter (as its name indicates). It is present in the bacterial uptake and maintenance process, so it influences the bacterial growth process. Its effect is shown in figure 7.26. A higher value in *yield* causes the bacteria to uptake more nutrient and to obtain more energy per unit of nutrient, this results in an increase in the biomass concentration and the viability of the population (that dies later than in the nominal case). For a small value in *yield*, the effect is the opposite, as expected.

This coefficient *yield* also influences the bacterial decay process (figure 7.26a), as the equivalent Monod parameter Y (as it was observed in figure 7.10).

If we want to modify the decay rate of bacteria, we can also manipulate

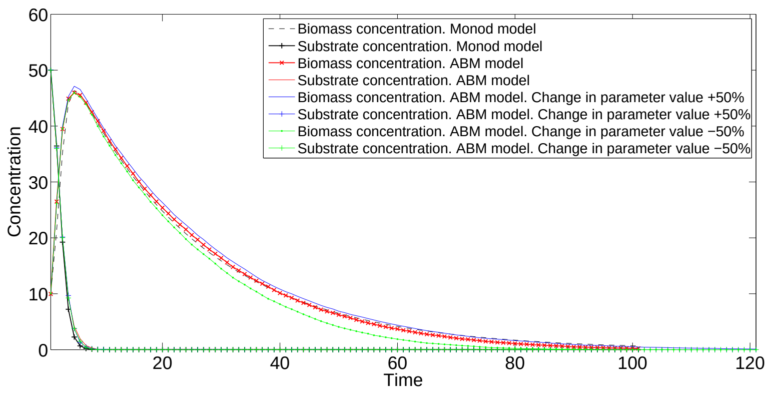


Figure 7.27: Changes in decay rate of bacteria modifying *viability* parameter from the ABM.

the *viability* parameter, that is, the mean value of bacterial deaths. So, the higher the *viability* parameter, the older the bacteria would die. This effect is shown in figure 7.27.

The responses in general are similar; they have the same behaviour as expected. The ABM seems to be more sensitive to variations, in general.

7.4.1.4 Studying the influence of bacterial composition in the dynamics of the system

The most common use of this type of ABM models is for understanding the behaviour of the model under different hypothesis than cannot be tested in the real system (availability, cost, etc). In this thesis we have tested the influence of the composition of the bacterial culture in the reactor, that is, the viability and size of the bacteria inside the reactor, and in the input influent stream. These experiments have been done for the continuous operating protocol. The ABM is parametrized with the values obtained in the subsection 7.4.1.2.

We analysed four different scenarios, four bacterial compositions of the influent stream. We have named each scenario with labels (*in italics*), which also appear as legends in the figures of this section.

- *Viable biomass input influent.* The bacteria in the influent are young and viable, and their mass are near the reproduction mass. In the figures of the experiments it is represented in blue.

- *Like steady state bacteria input influent.* The bacteria in the influent are a mixture between viable and less viable bacteria, of different sizes, with the same composition as the bacteria inside the reactor when the system is on steady state. This bacterial composition is obtained from a previous simulation (parametrized as table 7.8) collecting a sample that is used later for 'cloning' new bacteria for the influent. In the figures of the experiments it is represented in green.

- *Suboptimal bacteria input influent.* The bacteria are in a suboptimal viability condition, and their mass are near the reproduction mass. In the figures of the experiments it is represented in pink.

- *50% viable and 50% mean biomass input influent.* In this case, half of the bacteria in the influent are equal than in the second case, and the other half has a composition equal to the average values of the bacteria inside the reactor when the system is on steady state. This second half is obtained from a previous simulation (parametrized as table 7.8), measuring the population mean characteristics (size, viability, etc) and using these values to randomly generate new bacteria with properties as a normal distribution centred in the mean calculated values. In the figures of the experiments it is represented in brown.

We compared the scenarios against the prediction of the Monod continuous model behaviour (equations 7.1 and 7.2). The parameter values for the Monod model were the ones in table 7.7 and $V = 50m^3$, $q = 1m^3/s$, these two last values are not real values, like the rest of them, only arbitrary values for simulation. The parameters value for the ABM are the ones in table 7.8 except for $flow = 0.6$, $X0 = 10$, $S0 = 50$.

$$\frac{dX}{dt} = \mu X - K_d X + \frac{q}{V}(X_0 - X) \tag{7.1}$$

$$\frac{dS}{dt} = -\frac{\mu}{Y} X + \frac{q}{V}(S_0 - S) \tag{7.2}$$

where X is the biomass concentration, μ is the specific growth rate and K_d is the decay coefficient, q is the flow in the influent and in the effluent, V is the reactor volume, X_0 is the biomass concentration in the influent, X is the biomass concentration in the effluent, Y is the yield coefficient, S_0 is the substrate concentration in the influent, and S is the substrate concentration in the effluent.

The first experiment tests the influence of the bacterial compositions in the influent in the dynamics of the system. The four scenarios are compared with the Monod Model prediction. The simulation starts with concentrations of biomass and substrate inside the reactor $X0i = 92$ and $S0i = 14$ respectively, and with viable bacteria which mass is around the reproduction mass. The results are shown in figure 7.28. As it can be observed, the composition of the bacterial culture in the influent strongly affects the steady state the system reaches.

The next experiment consisted in modifying the concentration of substrate in the influent (an increment of 20% and a decrement of 20%) at time=10 and then maintaining this new concentration constant (simulating for example an increment of substrate in the sewage). As figure 7.29 shows, the dynamics of the system for the different scenarios show that the systems reaches different steady states. This implies that the system is working in a different point depending on the influent, and should be controlled with the recirculation of bacteria of the activated sludge process, which is important in a real system from a practical point of view. We have obtained analogous results when the change is performed in the concentration of biomass in the influent (figure 7.30), and in the influent and effluent flow (since it is the same in this model) (figure 7.31).

7.4.2 Conclusions

In this work we have developed and implemented an ABM for the activated sludge process, which provides the possibility to test the behaviour of the system under different operating protocols and also the influence of the bacterial culture composition.

We can conclude that the bacterial culture composition definitely influences the dynamics of the system. That influence is present in the recirculation of sludge to the tank and in the addition of new bacterial material in

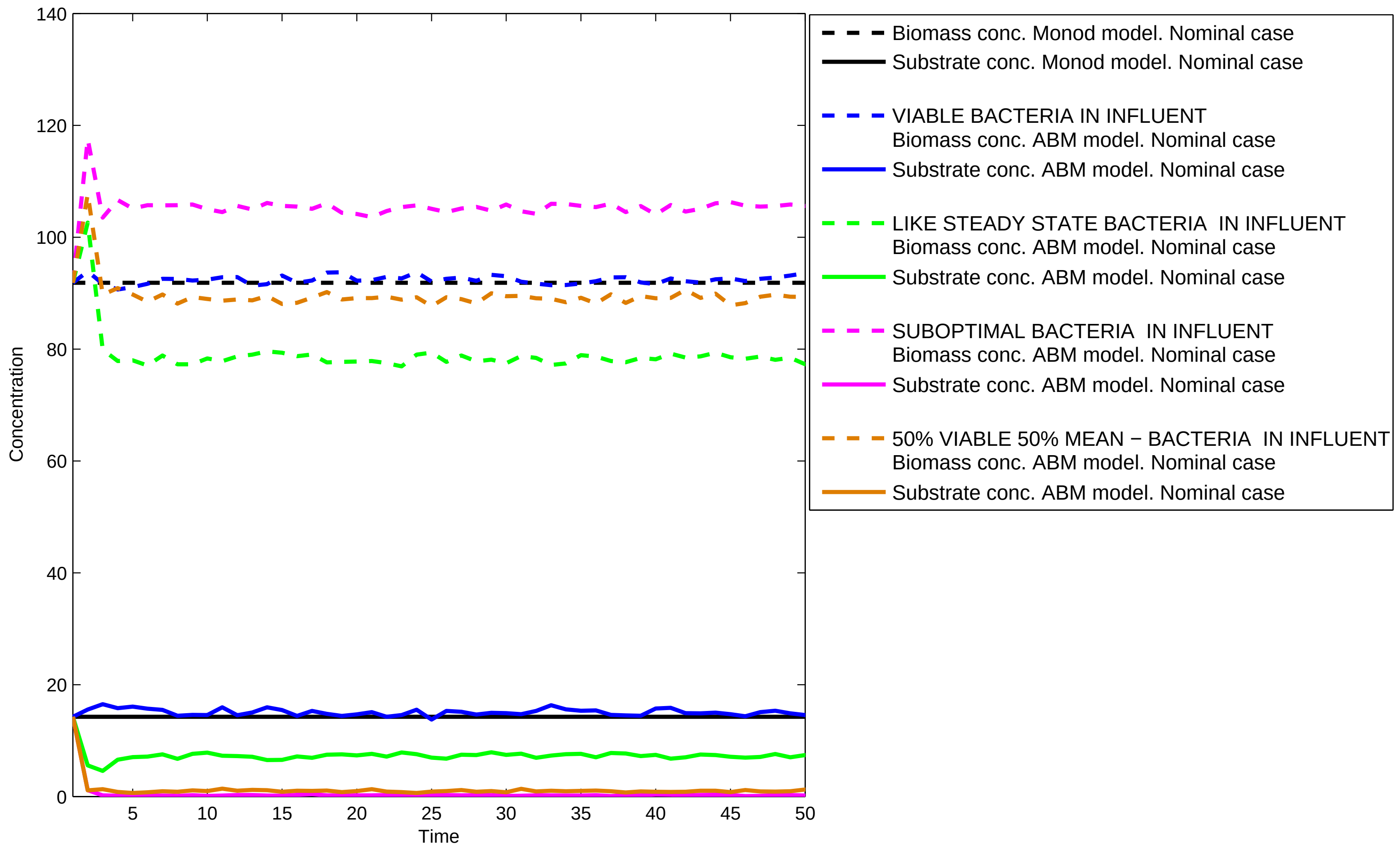
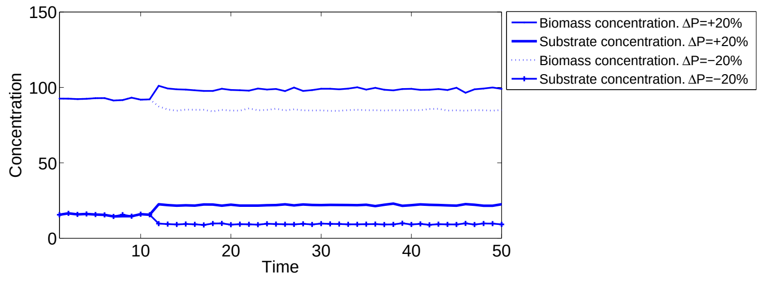
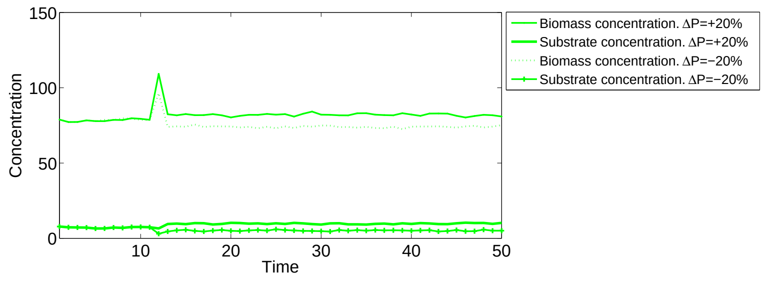


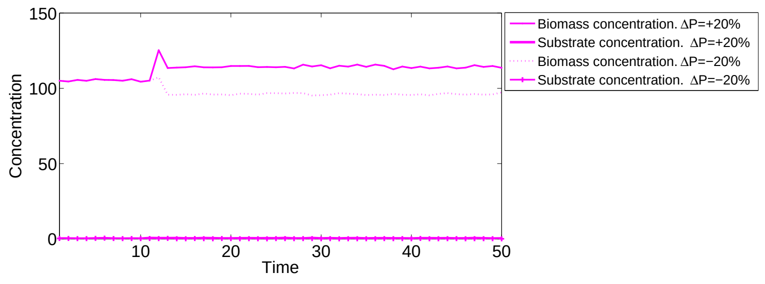
Figure 7.28: Comparison of the steady state of the system with different bacterial compositions of the input influent.



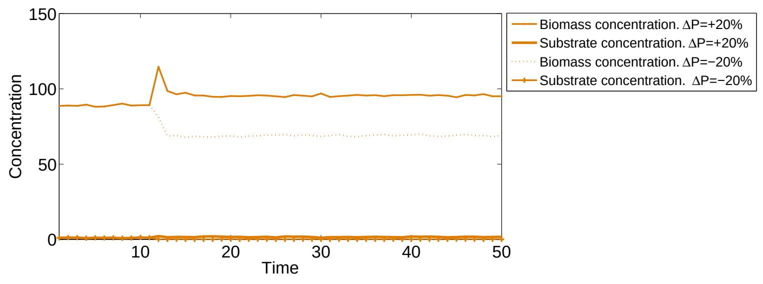
(a) *Viable biomass input influent.*



(b) *Like steady state bacteria input influent.*

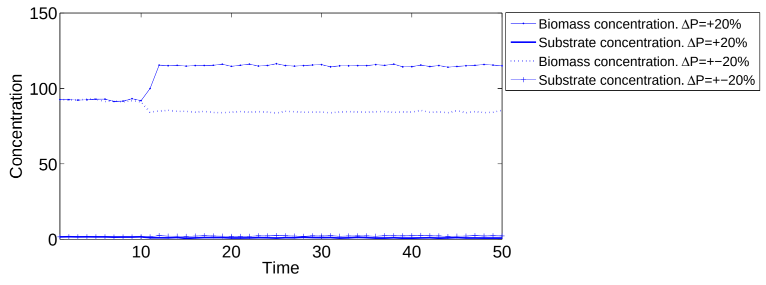


(c) *Suboptimal bacteria input influent.*

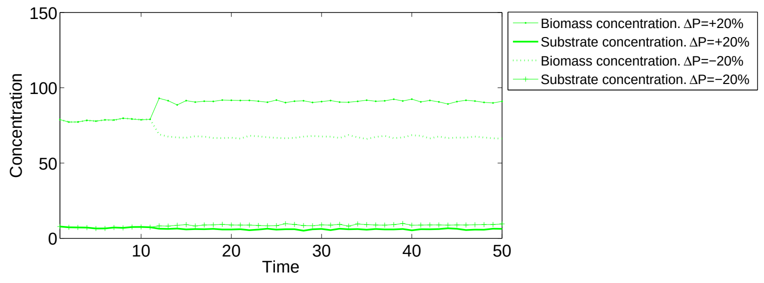


(d) *50% viable and 50% mean biomass input influent.*

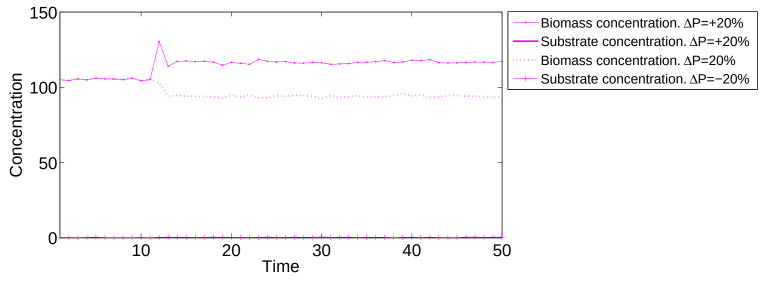
Figure 7.29: Dynamic behaviours of the four scenarios when a change in the concentration of substrate in the influent is conducted.



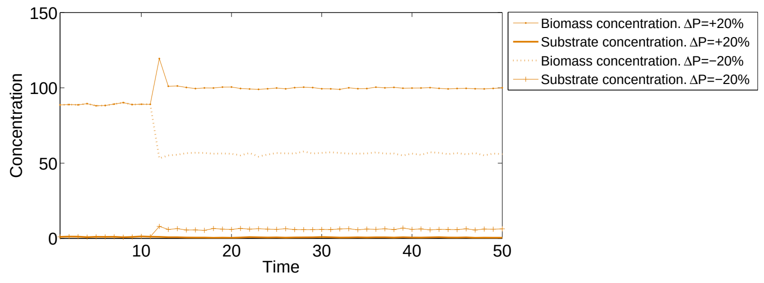
(a) *Viable biomass input influent.*



(b) *Like steady state bacteria input influent.*

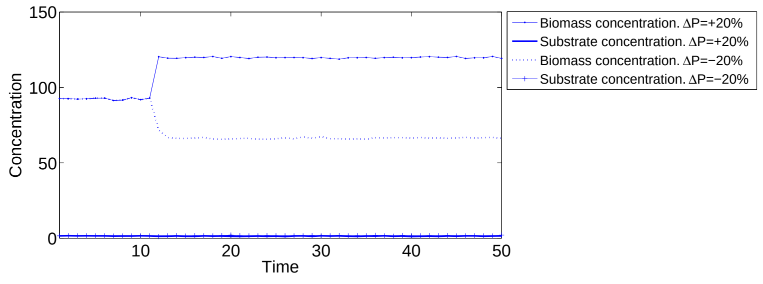


(c) *Suboptimal bacteria input influent.*

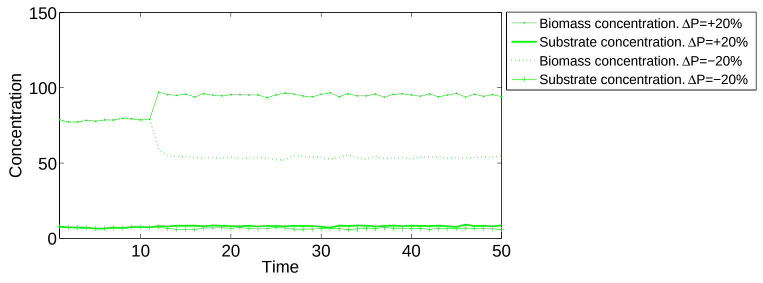


(d) *50% viable and 50% mean biomass input influent.*

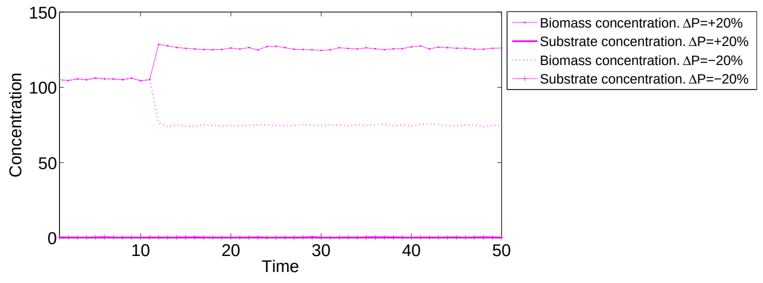
Figure 7.30: Dynamic behaviours of the four scenarios when a change in the concentration of biomass in the influent is performed.



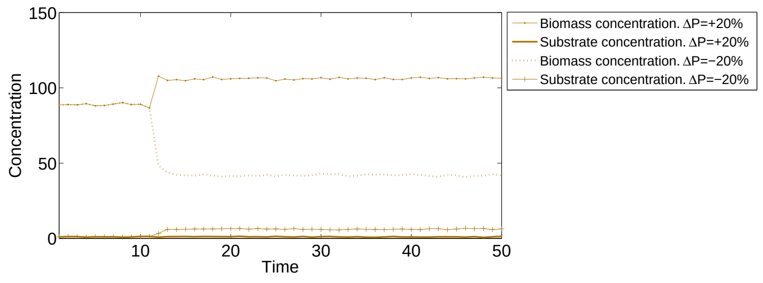
(a) *Viable biomass input influent.*



(b) *Like steady state bacteria input influent.*



(c) *Suboptimal bacteria input influent.*



(d) *50% viable and 50% mean biomass input influent.*

Figure 7.31: Dynamic behaviours of the four scenarios when a change in the influent flow is conducted.

the activated sludge process. This information will be very valuable for the optimal control of the process.

We adjusted the model parameters for the batch operation protocol with the objective to mimic the Monod model as if it were a real system. We have experienced the challenging of this process of tuning, which is more complicated when more and more parameters are involved. The general way to proceed is adjusting the majority of parameters manually with real values (since some individual properties are also real properties), and using an optimization algorithm to adjust the rest, if necessary.

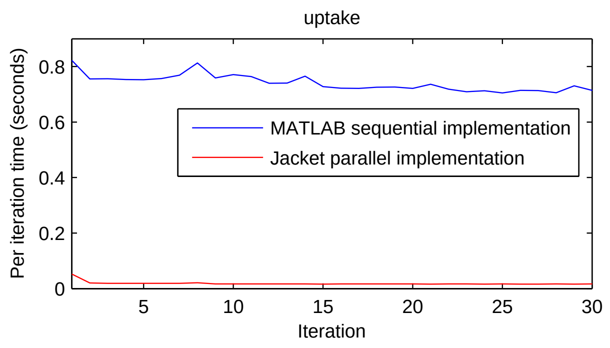
This model is available for download online at `https://csp.isa.cie.uva.es/~mpereda/ABM_SWARM_THESIS/`.

7.5 CUDA implementations

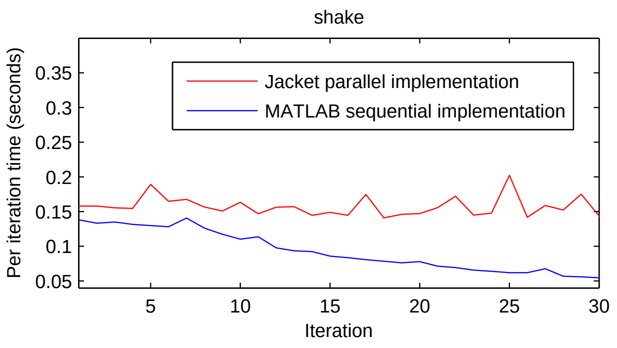
When parallel computing over GPU is mixed with ABM, one can be tempted to parallelize the complete ABM, since it is composed by quasi independent agents. Moreover, one disadvantage of using agent-based modelling is that in many cases the models have high computational load. Our aim in this part is applying GPGPU to try to reduce the computing time of the ABM simulations.

We parallelized our agent-based model (section 7.3), obtaining quite disappointing results. First, we needed a lot of shared memory between threads, because, although the individuals in our model are independent, the actions one individual performs indirectly influence the others (for example in the uptake process). This implies a first hardware limitation, since the shared memory in these devices has generally a small capacity or the access is very slow to perform many accesses in each iteration.

We performed three implementations to prove this point. The first was CUDA native code, where we run out of shared memory and could not finish the implementation.



(a) Uptake method



(b) Reactor stirring method

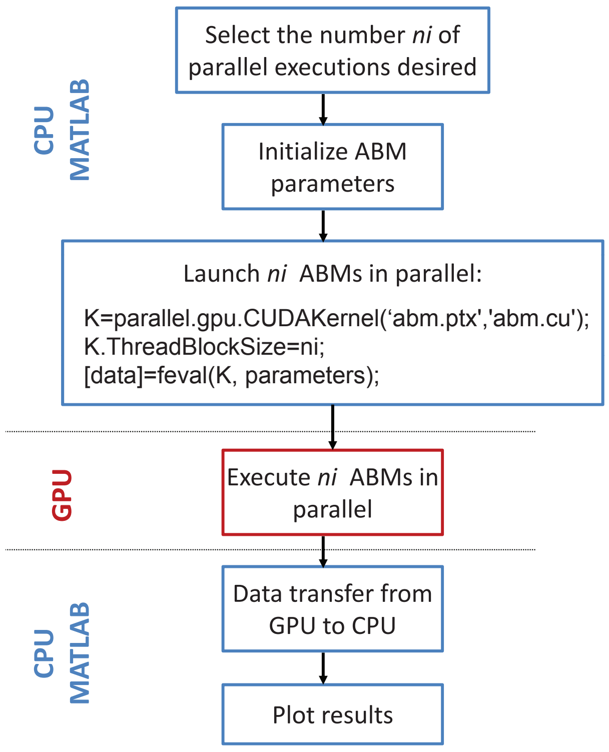
Figure 7.32: Comparison of sequential and *Jacket* implementations.

Afterwards, we decided to use the MATLAB *Parallel Computing toolbox*. We managed to adapt our code to solve the many limitations this toolbox forces you to comply (no indexing allowed in parallel code, no nested *for* loops, no *if* statements inside *for* loops...), and we also had to share back data from GPU to MATLAB workspace many times which made the implementation very inefficient. As a result, we had an implementation that was slower than the MATLAB original implementation.

The last option we tried was the *Jacket* toolbox for MATLAB (from *Accelereyes* [3]) (nowadays this company has joined MATLAB to develop an unique toolbox for MATLAB, the *Parallel Computing Toolbox* as we can read in the blog from *Accelereyes* [2]). This toolbox was less restrictive and the programming experience was better since we had not to adapt our code so much. But the results were not satisfactory enough; our parallel implementation was sometimes slower and sometimes faster, but it does not worth the effort involved in programming, as we can see in figure 7.32 for the uptake and reactor stirring methods.

Then we decided to implement our ABM as a CUDA kernel and provide the possibility to run several simulations in parallel, decreasing the computational time of the process of tuning of parameters and also providing a fast option to test different hypotheses in parallel. The kernel was launched with MATLAB, this process is summarized in figure 7.33. Firstly, the number of parallel simulations to be launched must be selected, and also the parametrization for each simulation. Then, with the *Parallel Computing Toolbox from MATLAB*, we created a CUDA kernel object K, we selected launching *ni* parallel simulations; and at last, we launched the simulations with the *feval* MATLAB function. This function copies back the data from the GPU to the MATLAB workspace, so we can plot the simulation results.

In figure 7.34 we can observe fifty parallel simulations of our ABM, all the simulations with the same parameters, so we can observe the influence of the stochasticity in our model. These fifty simulations were executed in parallel taking 3.63 seconds. If we compare it with 7.97 seconds that was required in an ANSI C implementation, we can see an improvement of 2x in execution time.


CPU
MATLAB
Select the number *ni* of parallel executions desired
Initialize ABM parameters
Launch *ni* ABMs in parallel:
GPU
Execute *ni* ABMs in parallel
CPU
MATLAB
Data transfer from GPU to CPU
Plot results

```
K=parallel.gpu.CUDAKernel('abm.ptx','abm.cu');
K.ThreadBlockSize=ni;
[data]=feval(K, parameters);
```

Figure 7.33: Parallel launch scheduling

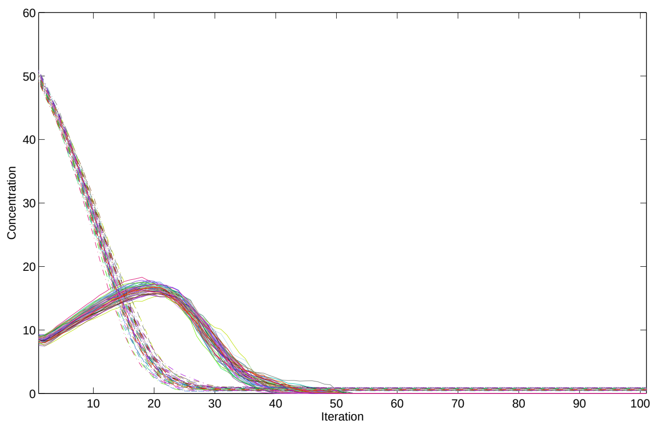


Figure 7.34: 50 Parallel ABM simulations.

In the next chapter...

In the next section, our swarm intelligence optimization algorithm, the Hiker, is introduced.

Chapter 8

Swarm Intelligence: The Hiker algorithm

Things don't have to change the world to be important.

Steve Jobs.

Summary: This chapter exposes our work developing a new optimization algorithm called *Hiker*, under the *swarm intelligence* paradigm. The algorithm has been implemented in MATLAB and its performance compared on several benchmark problems.

8.1 The Hiker optimization algorithm

The Hiker optimization algorithm is inspired by the swarm intelligence theory and the particle swarm algorithm. The algorithm represents the collective behaviour of a group of hikers descending a mountain, thus representing a minimization problem. The hikers carry walkie talkies to share information among them.

Every individual (*hiker*) in the swarm represents a potential candidate solution and it is represented by a multidimensional (n_x) position vector. The algorithm determines the positions each hiker could explore. The new position each hiker selects is influenced by the best position the entire swarm has found. At each iteration, the hiker has the possibility to explore nd random directions and p points in every direction, emulating the vision that a man can have descending a mountain. The points in every direction are calculated with an update formula (equation 8.1) based on the Swann bracketing algorithm [57], as it is shown in figure 8.1. The set of new potential positions of every hiker is also influenced by the past position with a inertia coefficient γ.

$$\vec{x_{p_i}}(t) = \vec{x_{0i}}(t) + 2^j r_{min}\vec{v}_d + \gamma(\vec{x_{0i}}(t) - \vec{x_{0i}}(t-1)) \tag{8.1}$$

where $\vec{x_{0i}}(t)$ denote the position of hiker i in the search space at time step t; unless otherwise stated, t denotes discrete time steps. $\vec{x_{p_i}}(t)$ denote the new possible position at direction d (from 1 to nd). $\vec{v}_d$ is the random direction vector. j goes from 1 to p points. $rmin$ is an adaptive resolution coefficient depending on the variation of the objective function of every hiker i (figure 8.2), $ratio$ is calculated as shown in equation 8.2:

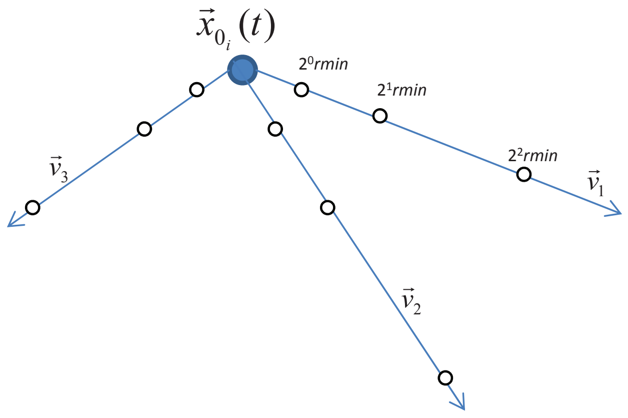


Figure 8.1: Sketch of a hiker generating possible points in three directions.

$$ratio = |f(\vec{x_{0i}}(t)) - f(\vec{x_{0i}}(t-1))| \tag{8.2}$$

where $f : \mathbb{R}^{n_x} \to \mathbb{R}$ is the fitness function, which measures the quality of the solution found.

Every hiker i selects as his next possible position the best from the $nd \times p$ potential positions (equation 8.3).

$$\vec{x_{p_i}}(t) \in \left\{\vec{x_{p_{i_1}}}(t), ..., \vec{x_{p_{i_{nd\times p}}}}(t)\right\} |f(\vec{x_{p_i}}(t)) = min\left\{f(\vec{x_{p_{i_1}}}(t)), ..., f(\vec{x_{p_{i_{nd\times p}}}}(t))\right\} \tag{8.3}$$

The best potential position in the swarm is selected from the particles of the current swarm

$$\widehat{\vec{x_p}(t)} \in \left\{\vec{x_{p1}}(t), ..., \vec{x_{pni}}(t)\right\} |f(\widehat{\vec{x_p}(t)}) = min\left\{f(\vec{x_{p1}}(t)), ..., f(\vec{x_{pni}}(t))\right\} \tag{8.4}$$

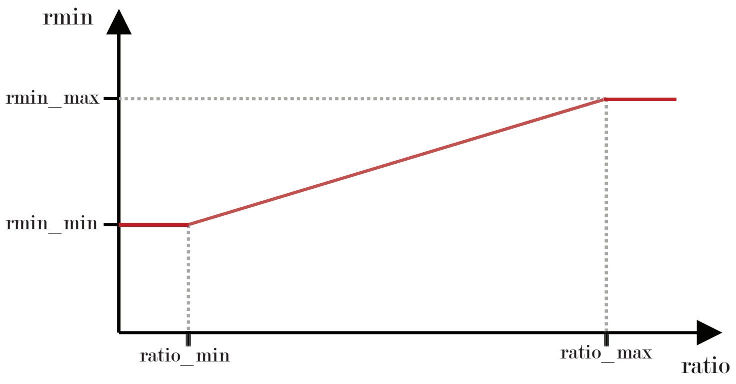


Figure 8.2: rmin value in function of ratio value.

where n_i is the number of hikers.

The contribution to each hiker's position of the global best is weighted with the parameter α. A hiker only changes its position to a new one if the fitness function improves; considering a minimization problem we have equation 8.5:

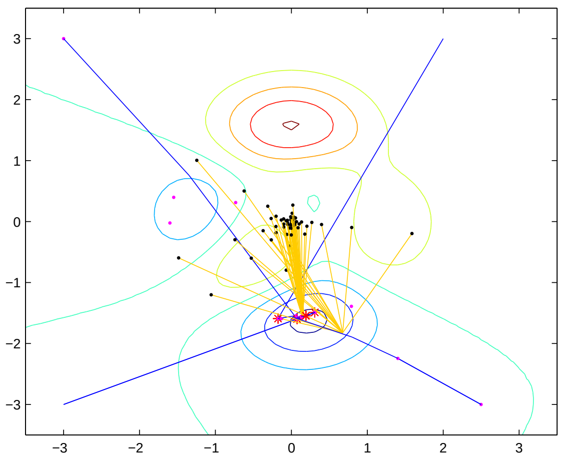
$$\vec{x_0}_i(t+1) = \begin{cases} \vec{x_0}_i(t) & if\ f((1-\alpha)\cdot\vec{x}_{p_i}(t) + \alpha\cdot\widehat{\vec{x_p}(t)}) \geq f(\vec{x_0}_i(t)) \\ (1-\alpha)\cdot\vec{x}_{p_i}(t) + \alpha\cdot\widehat{\vec{x_p}(t)} & if\ f((1-\alpha)\cdot\vec{x}_{p_i}(t) + \alpha\cdot\widehat{\vec{x_p}(t)}) \\ & < f(\vec{x_0}_i(t)) \end{cases} \tag{8.5}$$

The Hiker method is summarized in Algorithm 2

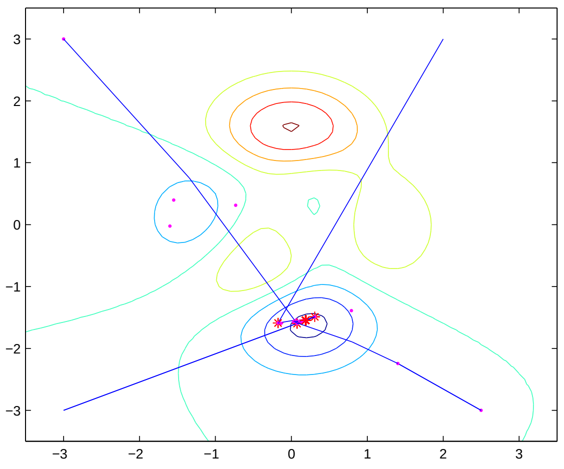
Algorithm 2 *Hiker*

```
Create and initialize an $n_x$−dimensional swarm;
repeat
  for each hiker $i = 1, ..., n_i$ do
    create $nd$ random directions
    create the possible positions with eq. 8.1
    select the best possible position with eq. 8.3
    set the global best position with eq. 8.4
    update the position using equation 8.5
  end for
until stopping condition is true;
```

In figure 8.3 a representation of the optimization process of four hikers optimizing the function Peaks (equation 8.10) can observed. The trail of the four hikers after the optimization appears in the figure in dark blue, the new possible route in yellow, in black the new possible end points, in magenta the best point every hiker found in past iterations, in red the best point every hiker has found in current iteration.



(a) Hiker optimization process with four hikers.



(b) Trail of four hikers after an optimization process.

Figure 8.3: Representation of the Hiker algorithm during the optimization of Peaks function.

8.2 Implementation Aspects

The algorithm ends when a stopping condition is satisfied. We have implemented three conditions, so the optimization process ends if one of them is satisfied:

- Terminate when an acceptable solution has been found $|f(\widehat{\vec{x_p}(t)})| < fobj$.
- Terminate when the hiker in the current better solution has not observed improvement after a certain number *max_counter* of consecutive iterations.
- Terminate when a maximum number of iterations N has been reached.

Other stopping conditions can also be implemented, such as the ones explained at page 34 in section 3.2.2.

With respect to the initialization process, the variables that have to be initialized are:

- The number of hikers n_i.
- The maximum and minimum values for ratio (*ratio_min*, *ratio_max*) and rmin (*rmin_min*, *rmin_max*).
- The minimum variation of the objective function for not considering a hiker stuck at a certain position (*ratio_end*). The maximum number of iterations the hiker in the better value found can be stuck (*max_counter*).
- The value for an acceptable solution $fobj$.
- The contribution of the global best (α).
- The inertia coefficient γ.
- The maximum number of iterations.
- The number of directions every hiker explores (nd) and the number of points (p) in every direction.

8.3 Results

We have implemented Hiker in MATLAB, parallelizing some parts of the code with the *Parallel Computing Toolbox* for MATLAB. We have tested the efficiency of the proposed algorithm on a set of benchmarck optimization functions.

The parallelized code has been implemented for GPU computing. The GPU used is a NVIDIA GeForce GTX 480. The machine used for running the algorithms is an Intel Core i5 650 3.20GHz, 8GB DDR3.

We have optimized five different benchmark functions, four from [26]: spherical, griewank, hiperellipsoid, rosenbrock (in equations 8.6 - 8.9, n_x is the dimension of the problem); and one from MATLAB: peaks.

Spherical:

$$f(x) = \sum_{j=1}^{n_x} x_j^2 \tag{8.6}$$

with $x_j \in [-100, 100]$ and $f^* = 0.0$.

Griewank:

$$f(x) = 1 + \frac{1}{4000}\sum_{j=1}^{n_x} x_j^2 - \prod_{j=1}^{n_x} \cos\left(\frac{x_j}{\sqrt{j}}\right) \tag{8.7}$$

with $x_j \in [-600, 600]$ and $f^* = 0.0$.

Hiperellipsoid:

$$f(x) = \sum_{j=1}^{n_x} j^2 x_j^2 \tag{8.8}$$

with $x_j \in [-1, 1]$ and $f^* = 0.0$.

Rosenbrock:

$$f(x) = \sum_{j=1}^{n_x-1} \left[(1 - x_j)^2 + 100(x_{j+1} - x_j^2)^2\right] \tag{8.9}$$

with $x_j \in [-2.048, 2.048]$ and $f^* = 0.0$.

Peaks:

$$\begin{aligned} f(x,y) = {} & 3(1-x)^2 \exp(-x^2 - (y+1)^2) - 10(x/5 - x^3 - y^5)\exp(-x^2 - y^2) \\ & -1/3\exp(-(x+1)^2 - y^2) \end{aligned} \tag{8.10}$$

with $f^* = -6.5511$ in the point (0.2283,-1.6255).

8.3.1 Comparison of implementations in MATLAB

The first experiments compare the performance between the MATLAB implementation of PSO and Hiker, where parallelization could take place in the function evaluation for every particle (hiker), as algorithm 3 shows for the Hiker algorithm.

The parametrization for the PSO is shown at table 8.1, and for the Hiker is shown at table 8.2.

Parameter	Value	Parameter	Value
$ratio\_end$	0	ω	0.9
K_1	0.3	K_2	0.3
n_s	10	N	200

Table 8.1: PSO parametrization.

Parameter	Value	Parameter	Value
$rmin\_min$	0.001	min_fobj	-10
$rmin\_max$	0.5	α	0.5
$ratio\_min$	0.01	γ	0.2
$ratio\_max$	10	N	200
$ratio\_end$	0.00001	nd	10
$max\_counter$	10	p	4
ni	10		

Table 8.2: Hiker parametrization.

The two algorithms find the minimum for the five functions with 2 dimensions, as table 8.3 shows. Table 8.4 shows the stop iteration for each optimization. The execution time of both algorithms is similar (table 8.5), in general.

	PSO	Hiker
Peaks	-6.5511	-6.5511
Spherical	3.17E-09	5.50E-13
Griewank	4.08E-05	2.88E-08
Hiperhellipsoid	1.68E-11	1.15E-07
Rosenbrock	5.77E-05	0.005

Table 8.3: Solutions found.

	PSO	Hiker
Peaks	200	200
Spherical	200	16
Griewank	200	36
Hiperhellipsoid	200	60
Rosenbrock	200	200

Table 8.4: Stop iteration of the optimization process of several functions.

	PSO	Hiker
Peaks	0.08	0.11
Spherical	0.07	0.09
Griewank	0.09	0.13
Hiperhellipsoid	0.07	0.48
Rosenbrock	0.1	0.48

Table 8.5: Execution times of the optimization of several functions.

Algorithm 3 *Hiker pseudocode*

```
CPU: create and initialize an $n_x$−dimensional swarm;
repeat
  for each hiker $i = 1, ..., n_i$ do
    CPU: create $nd$ random directions
    CPU: create the possible positions
    GPU: evaluate the objective function on the possible positions
    CPU: select the best possible position
    CPU: set the global best position
    CPU: update the position of the hiker
  end for
until stopping condition is true;
```

8.4 Conclusions

We have proposed an optimization algorithm inspired by the swarm intelligence theory. We have tested the algorithm on some benchmark problems, and results in terms of performance are promising.

We implemented both the Hiker and the PSO algorithms in parallel with the *Parallel Computing Toolbox* from MATLAB, but we obtained quite disappointing results in terms of execution time for both algorithms as compared with serial implementations; 10x worst in case of PSO GPU and 100x in case of Hiker GPU in which the number of data transferred between CPU and GPU is much greater ($nd \times p$ per each hiker). This was partly caused by the inefficient memory transferences from CPU to GPU and vice versa, and also due to the programming limitations this toolbox presented by the time we developed this work.

This algorithm is available for download online at `https://csp.isa.cie.uva.es/~mpereda/ABM_SWARM_THESIS/`.

Part IV

Conclusions and Future Work

Chapter 9

Conclusions and Future Work

Try to learn something about everything
and everything about something.
Thomas Henry Huxley

Summary: In this chaper we present the conclusions of the work presented in this dissertation, the main contributions and the future work directions.

9.1 Conclusions

In this thesis we have worked on bringing together the system engineering and automatic control field and the agent-based modelling and swarm intelligence techniques. We have studied the activated sludge process from the point of view of its microscopic constituents. The conclusions resulting from this work are:

- We have summarize the ABM methodology steps so that a newcomer could use the Fundamentals chapter 2 as initial guide on the subject.
- We have proposed a new notation SSABR for representing agent-based models, which is closer to the systems engineering field and points out the dynamic nature of agents' behaviour through a state space representation. Examples of application have been included to illustrate the utility and application of the SSABR. This notation could be used to teach ABM in the system engineering and automatic control field and also to report ABM research results, together with the ODD protocol.
- We have developed an ABM for the activated sludge process, that allows testing the influence of different reactor operating protocols and

a configurable bacteria population. Although the model can capture some of the macroscopic behaviours of the modelled process, it is indeed a simple version of the processes involved in the system. A single population of microorganisms and one type of substrate have been modelled, but in the real system several microorganisms and types of substrate are involved. The objective of this work is not to obtain a complete complex model but to analyze and confirm the suitability of this modelling technique to the systems engineering and automatic control field.

- We have confirmed the significant influence the oscillations in the influent and the variations in its composition have on an activated sludge process. Our model allows testing different scenarios to study the system dynamics.
- We have provided the possibility to launch hundreds of simulations in parallel, to study, for example, the influence of stochasticity of bacterial population in the macroscopic behaviour of the system.
- We have developed an optimization algorithm, the Hiker, inspired by the swarm intelligence theory with promising results of its performance.

9.2 Summary of contributions

- M. Pereda and J.M. Zamarreño. Agent-based modeling of an activated sludge process in a batch reactor. In 2011 19th Mediterranean Conference on Control & Automation (MED), pages 1128 - 1133, Corfu, June 2011. IEEE. [100].

  The aim of this work is to study the feasibility of using agent-based modelling to study the activated sludge process. A model in NetLogo has been proposed, and experiments have been developed comparing the model behaviour with a classical modelling approximation for this process, the Monod model.

- M. Pereda and J. M. Zamarreño. Modelado basado en agentes. In Actas de las XXXII Jornadas de Automática, pages 423 - 428. Universidad de Sevilla y CEAIFAC, Septiembre 2011. [101].

  The aim of this work is to study and adapt the agent-based model methodology to the field of industrial processes, to allow the representation of macroscopic behaviours of the processes from the modelling of its microscopic constituents. This work presents an introduction to the subject.

- M. Pereda and J. M. Zamarreño. An OOP agent-based model for the activated sludge process using MATLAB. In S. Omatu, J. F. De Paz

Santana, S. R. González, J. M. Molina, A. M. Bernardos, and J. M. C. Rodríguez, editors, Distributed Computing and Artificial Intelligence, volume 151 of Advances in Intelligent and Soft Computing, pages 241 - 248. Springer Berlin Heidelberg, 2012. [102].

This contribution showed that agent-based modelling allows the researcher to develop a bottom-up approximation to the study of complex problems such as the phenomena occurring in a wastewater treatment plant, and in particular, secondary treatment by activated sludge. An Object Oriented Agent-Based Model for the activated sludge process in a batch reactor was proposed, which allows to analysing and a better understanding of the phenomena that occur in the system while varying some of its characteristics. The implementation environment chosen for the model, that is, MATLAB, allows an efficient representation and provides good design tools.

Other works published during the developing of this dissertation are:

- M. Pereda and J. M. Zamarreño. Optimización multiarranque en paralelo sobre GPU. In R. S. Matías García, editor, Actas de las XXXIII Jornadas de Automática, pages 423 - 428. Universidad de Vigo y CEAIFAC, Septiembre 2012. [103].

  Some control techniques require the optimization of an objective function to calculate the control action. Usually it implies a problem of high computational cost so some of these control techniques cannot be applied in real time. Code parallelization is giving very good results to decrease the execution time of implementations in many fields. This paper aims to prove the enormous potential of code parallelization techniques applied to solving optimization problems, in this case, multistart local search algorithms. This work is presented in appendix B.

9.3 Future directions

To continue these four years work, many research lines can be proposed. We summarize the ones we consider more interesting or plausible:

- To apply agent-based modelling to other systems, following the application advices in section 2.3.
- To extend the ABM for the activated sludge process to include other phenomena that could influence the dynamics, such as population of other microorganisms, the configuration of different types of substrate, the influence of the clarifier, the influence of some substrate components as nitrates and phosphates, etc and also improve the model of the bacteria.

- To use real data that allows the parametrization and calibration of the model.
- To integrate the ABM for the activated sludge process as a submodel in a whole wastewater treatment plant model.
- To use the proposed ABM to perform more simulations and test new composition hypothesis.
- To propose an optimal parametrization for the Hiker parameters for certain benchmark problems.
- To integrate the Hiker optimizer and the proposed model into a control structure, such as the Model Based Predictive Control, taking into account the dynamic nature of the problem.
- To explore the possibility of adding some kind of basic Artificial Intelligence, such as Artificial Neural Networks, to the agents' behaviour.

Part V

Appendices

Appendix A

SSABR application examples

Summary: In this section we present a collection of simple ABM examples described using the SSABR notation proposed in this thesis, explained at chapter 6.

A.1 Hilltopping behaviour in butterflies

This model is an introductory example in the book *Agent-Based and Individual Based Modelling - A practical introduction* [106]. The model is developed by Guy Pe'er et al. [99] in 1999, and models the hilltopping behaviour in butterflies in which males and virgin or multiple-mating females seek topographic summits for the purpose of mating. The objective of the model is to analyse the formation of virtual corridors in the case of hilltopping butterflies in topographically heterogeneous landscapes.

Applying SSABR, we define two types of agents, the butterflies and the patches. The butterflies are mobile agents defined by two spatial states, their coordinates in a two dimensional environment ($x\_coordinate$, $y\_coordinate$). The butterflies also have a parameter governing its behaviour, called q: the probability of moving uphill (table A.2). The other type of agents are the patches, environment grid cells defined by coordinates (x_patch, y_patch) and *elevation* parameters (table A.2). The patches also have a state *visited*? explained in table A.1. In this implementation, the butterfly coordinates are continuous variables but the patch ones are discrete.

Butterflies

$j = 1$, Butterflies

$\vec{x}^1_{i_1} \in \mathbb{R}^2 \; i_1 = 1, ..., n_1$

$\vec{x}^1_{i_1} = \begin{pmatrix} x\ coordinate \\ y\ coordinate \end{pmatrix}$

$X^1(t) = \left(\vec{x}^1_1 \,...\, \vec{x}^1_{n_1}\right)$; $X^1(t) \in M_{2\times n_1}(\mathbb{R})$

$X^1(t) = \begin{pmatrix} x^1_1(t) \\ x^1_2(t) \end{pmatrix}$; $x^1_1(t),\ ...,\ x^1_2(t) \in M_{1\times n_1}(\mathbb{R})$

$\vec{p}^1_1 = \vec{p}^1_2 = \ ...\ = \vec{p}^1_{n_1} \equiv \vec{p}^1$ All butterfly agents share the same set of parameters

$p^1 \in \mathbb{R}$

$p^1 = \left(\ prob-of-moving-uphill\ \right) = q$

Landscape patches

$j = 2$, Landscape patches

$\vec{x}^2_{i_2} \in \mathbb{R}\ i_2 = 1, ..., n_2$

$\vec{x}^2_{i_2} = \left(\ visited?\ \right)$

$X^2(t) = \left(\vec{x}^2_1 \,...\, \vec{x}^2_{n_2}\right)$; $X^2(t) \in M_{1\times n_2}(\mathbb{R})$

$\vec{p}^2_{i_2} \in \mathbb{R}^3\ i_2 = 1, ..., n_2$

$\vec{p}^2_{i_2} = \begin{pmatrix} x\ patch \\ y\ patch \\ elevation \end{pmatrix}$

$P^2 = \left(\vec{p}^2_1 \,...\, \vec{p}^2_{n_2}\right)$; $P^2 \in M_{3\times n_2}(\mathbb{R})$

$P^2 = \begin{pmatrix} p^2_1 \\ p^2_2 \\ p^2_3 \end{pmatrix}$; $p^2_1,\ ...,\ p^2_3 \in M_{1\times n_2}(\mathbb{R})$

Agent type	State	Description
Butterfly	*x coordinate*	x coordinate of the butterfly
Butterfly	*y coordinate*	y coordinate of the butterfly
Patch	*visited*?	true if any butterfly has been in that patch

Table A.1: Agent states

The environment

$\vec{\alpha} \in \mathbb{R}^2$

$\vec{\alpha} = \begin{pmatrix} number-of-width-cells \\ number-of-height-cells \end{pmatrix}$

Dynamic methods

In this model there are two methods. Method one represents the flight of the butterflies. Method 2 represents the updating of patches' *visited*? state.

Step 1:

Agent type	Parameter	Description
Butterfly	q	probability of a butterfly of moving to the neighbouring patch with the highest elevation
Patch	*x patch*	x coordinate of the centre of the patch
Patch	*y patch*	y coordinate of the centre of the patch
Patch	*elevation*	elevation of the landscape patch

Table A.2: Parameters

$Z^1 = X^1(t) \quad c_1 = n_1$
$Z^2 = X^2(t) \quad c_2 = n_2$

Step 2:

$$Z^1 = \begin{pmatrix} z_1^1 \\ z_2^1 \end{pmatrix} \; \bar{Z}^1 = \begin{pmatrix} \bar{z}_1^1 \\ \bar{z}_2^1 \end{pmatrix}$$

$z_1^1,\ ...,\ z_2^1 \in M_{1\times c_1}(\mathbb{R});\ \bar{z}_1^1,\ ...,\ \bar{z}_2^1 \in M_{1\times \bar{c}_1}(\mathbb{R})$

Method 1: At each time step, a butterfly could move upward (from $z_1^1,\ z_2^1$ to $\bar{z}_1^1,\ \bar{z}_2^1$) to the highest ($p_3^2$) neighbouring patch ($p_1^2,\ p_2^2$) with a probability q ($p^1$) or move randomly to a neighbouring patch with a probability 1 - q, inside the limits of the environment ($\vec{\alpha}$).

$$\begin{pmatrix} \bar{z}_1^1 \\ \bar{z}_2^1 \end{pmatrix} = F_{i_1,1}^1(z_1^1,\ z_2^1,\ p_1^2,\ p_2^2,\ p_3^2,\ p^1,\ \vec{\alpha})$$

Method 2: If there is a butterfly in a patch, the state *visited?* of the patch will change from false to true.

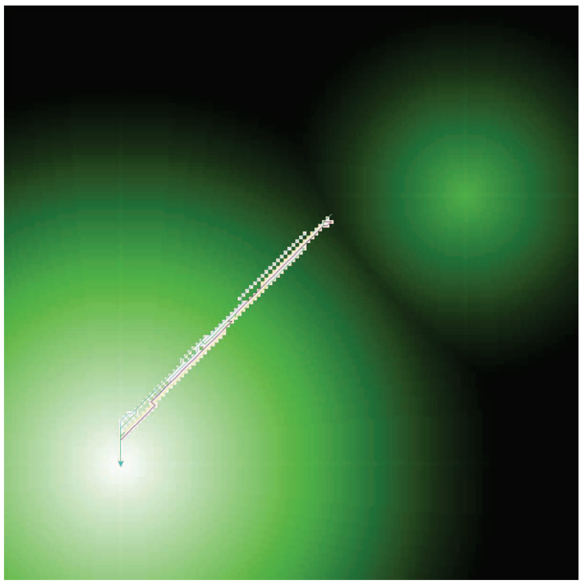
$$\begin{pmatrix} \bar{Z}^2 \end{pmatrix} = F_{i_2,1}^2(Z^2,\ z_1^1,\ z_2^1)$$

Step 3:

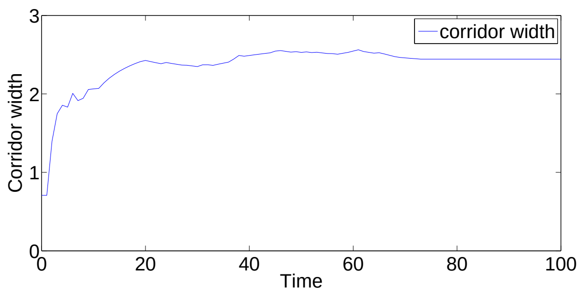
$X^1(t+1) = \bar{Z}^1 \quad n_1 = c_1$
$X^2(t+1) = \bar{Z}^2 \quad n_2 = c_2$

Aggregated variables

To estimate the strength of the channelling effect as a function of q, the number of patches visited by individuals (X^2) during a given simulation is counted and divided by the distance between the source coordinates ($x_1^1(0),\ x_2^1(0)$) (initialization value) and the summit ($x_1^1(t),\ x_2^1(t)$). This measure is called 'corridor width'. In this model there is only one aggregated



(a) Trail of four butterflies during the hilltoping. In white, the patches that have been visited.



(b) Evolution over time of the *corridor width*

Figure A.1: Representation of a simulation of the model.

variable measured, so $y(t) \in \mathbb{R}$.

$y(t) = g(X^2(t),\ x_1^1(0),\ x_2^1(0),\ x_1^1(t),\ x_2^1(t))$

A.1.1 Simulation example

We simulated the model with 4 butterflies and q=0.55 and the results are shown in Figure A.1. The visited patches are coloured in white. Every butterfly left a trail while moving. The dynamic evolution of the states is shown in figure A.1a, and the evolution of the aggregated variable *corridor width* is shown in figure A.1b.

A.2 Sugarscape 1 Immediate Growback

This model is a version from NetLogo library of examples (available at [130]) of Sugarscape model from Epstein and Axtell's, as described in chapter 2 of their book Growing Artificial Societies: Social Science from the Bottom Up [28] . It simulates a population with limited, spatially-distributed resources available.

Using SSABR we define two types of agents, the so called 'agents' which are the mobile agents, and the patches that are static agents. The 'agents' have three states: their coordinates in a two dimensional environment and the quantity of sugar they own (table A.3); and two parameters: *metabolism* and *vision* (table A.4). The patches are the grid cells defining the environment; they have three parameters: their grid coordinates (*x_coordinate*, *y_coordinate*) and the maximum sugar quantity they can hold (table A.4). They only have one state: the quantity of sugar they hold at time t (*psugar*).

Agents

$j = 1$, agents

$\vec{x}^1_{i_1} \in \mathbb{R}^3 \; i_1 = 1, ..., n_1(t)$

$$\vec{x}^1_{i_1} = \begin{pmatrix} x\ patch \\ y\ patch \\ sugar \end{pmatrix}$$

$$X^1(t) = \left(\vec{x}^1_1 \; ... \; \vec{x}^1_{n_1(t)}\right); \; X^1(t) \in M_{3\times n_1(t)}(\mathbb{R})$$

$$X^1(t) = \begin{pmatrix} x^1_1(t) \\ x^1_2(t) \\ x^1_3(t) \end{pmatrix}; \; x^1_1(t), \; ..., \; x^1_3(t) \in M_{1\times n_1(t)}(\mathbb{R})$$

$\vec{p}^1_{i_1} \in \mathbb{R}^2 \; i_1 = 1, ..., n_1(t)$

$$\vec{p}^1_{i_1} = \begin{pmatrix} metabolism \\ vision \end{pmatrix}$$

$$P^1 = \left(\vec{p}^1_1 \; ... \; \vec{p}^1_{n_1(t)}\right); \; P^1 \in M_{2\times n_1(t)}(\mathbb{R})$$

Patches

$j = 2$, Patches

$x^2_{i_2} \in \mathbb{R} \; i_2 = 1, ..., n_2$

$x^2_{i_2} = \left(\; psugar \; \right)$

$X^2 = \left(x^2_1 \; ... \; x^2_{n_2}\right); \; X^2 \in M_{1\times n_2(t)}(\mathbb{R})$

$\vec{p}^2_{i_2} \in \mathbb{R}^3 \; i_2 = 1, ..., n_2$

$$\vec{p}^2_{i_2} = \begin{pmatrix} x\ coordinate \\ y\ coordinate \\ max - psugar \end{pmatrix}$$

$$P^2 = \left(\vec{p}^2_1 \; ... \; \vec{p}^2_{n_2}\right); \; P^2 \in M_{3\times n_2}(\mathbb{R})$$

$$P^2 = \begin{pmatrix} p_1^2 \\ p_2^2 \\ p_3^2 \end{pmatrix};\ p_1^2,\ ...,\ p_3^2 \in M_{1\times n_2}(\mathbb{R})$$

Agent type	State	Description
Agent	*x patch*	x coordinate of the agent
Agent	*y patch*	y coordinate of the agent
Agent	*sugar*	the amount of sugar an agent has
Patch	*psugar*	the amount of sugar a patch contains

Table A.3: States

Agent type	Parameter	Description
Agent	*metabolism*	the amount of sugar that each agent loses each tick
Agent	*vision*	the distance that a agent can see in the horizontal and vertical directions
Patch	*x coordinate*	x coordinate of the centre of the patch
Patch	*y coordinate*	y coordinate of the centre of the patch
Patch	$max - psugar$	the maximum amount of sugar that a patch can hold

Table A.4: Parameters

The environment

$\vec{\alpha} \in \mathbb{R}^2$

$$\vec{\alpha} = \begin{pmatrix} number - of - width - cells \\ number - of - height - cells \end{pmatrix}$$

Dynamic methods

In this implementation there are two methods. The movement of 'agents', collection of sugar and die of 'agents' is implemented in method 1, and the growth of sugar of patches in method 2.

Step 1:

$Z^1 = X^1(t)\ \ c_1 = n_1(t)$

$Z^2 = X^2(t)\ \ c_2 = n_2$

In this model, the number of patches is fix, so n_2 is not time depending, but the number of agents is, so it is explicitly written as $n_1(t)$.

Step 2:

$$Z^1 = \begin{pmatrix} z_1^1 \\ z_2^1 \\ z_3^1 \end{pmatrix} \quad \bar{Z}^1 = \begin{pmatrix} \bar{z}_1^1 \\ \bar{z}_2^1 \\ \bar{z}_3^1 \end{pmatrix}$$

$z_1^1,\ ...,\ z_3^1 \in M_{1\times c_1}(\mathbb{R});\ \bar{z}_1^1,\ ...,\ \bar{z}_3^1 \in M_{1\times \bar{c}_1}(\mathbb{R})$

$$P^1 = \begin{pmatrix} p_1^1 \\ p_2^1 \end{pmatrix} \quad P^2 = \begin{pmatrix} p_1^2 \\ p_2^2 \\ p_3^2 \end{pmatrix}$$

$p_1^1,\ ...,\ p_2^1 \in M_{1\times q_1}(\mathbb{R});\ p_1^2,\ ...,\ p_3^2 \in M_{1\times q_2}(\mathbb{R})$

Method 1: Each agent can only see along a certain distance both horizontally and vertically. At each tick, each agent will move to the nearest unoccupied location (from $(z_1^1,\ z_2^1)$ to $(\bar{z}_1^1,\ \bar{z}_2^1)$) within their vision range (p_2^1) with the most sugar (Z^2), inside the limits of the environment $\vec{\alpha}$. If its current location has as much or more sugar (Z^2) than any unoccupied location it can see, it will stay put. After moving or not, each turtle will collect all the sugar (Z^2) in its patch ($p_1^2,\ p_2^2$), incrementing its sugar (from $z_3^1\ to\ \bar{z}_3^1$), decrementing the patch sugar to zero (from $Z^2\ to\ \bar{Z}^2$). Then, agents with sugar (z_3^1) less or equal to zero will die. Agents also use (and thus lose) a certain amount of sugar each tick (from $z_3^1\ to\ \bar{z}_3^1$), based on their metabolism rates (p_1^1).

$$\begin{pmatrix} \bar{Z}^1 \\ \bar{Z}^2 \end{pmatrix} = F_{i_1,3}^1(z_1^1,\ z_2^1,\ z_3^1,\ p_2^1,\ \vec{\alpha},\ Z^2,\ p_1^2,\ p_2^2,\ p_1^1)$$

Method 2: At each tick, each patch grows back fully to have the maximum amount of sugar (p_3^2).

$$\begin{pmatrix} \bar{Z}^2 \end{pmatrix} = F_{i_2}^2(Z^2,\ p_3^2)$$

Step 3:

$X^1(t+1) = \bar{Z}^1 \quad n_1(t+1) = \bar{c}_1$

$X^2(t+1) = \bar{Z}^2 \quad n_2 = \bar{c}_2$

Aggregated variables

In this implementation, four aggregated variables are measured ($\vec{y}(t) \in \mathbb{R}^4$).

The **population** is the number of 'agents' at time t.

$y_1(t) = g_1(X^1(t))$

The second one is the distribution of sugar inside 'agents' called **Wealth**

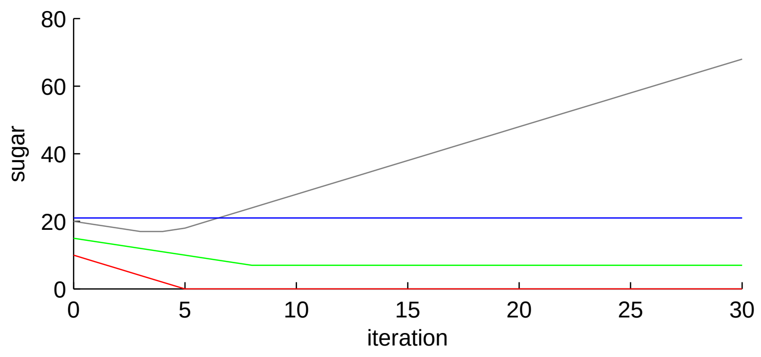


Figure A.2: Evolution of four turtles *sugar* states.

distribution.
$y_2(t) = g_2(x_3^1(t))$

The third one is the **Average vision**, that is, the mean of vision of 'agents'.
$y_3(t) = g_3(p_2^1)$

The **Average metabolism** is the mean of metabolism of 'agents'.
$y_4(t) = g_4(p_1^1)$

A.2.1 Simulation example

We simulated the model with 800 turtles, with random vision in a range 1-6 and random metabolism in a range 1-4. In figure A.2 the evolution of the quantity of sugar for four turtles is shown.

A.3 Particle Swarm Optimization Algorithm

This is the representation of Particle Swarm Optimization (PSO) algorithm with the SSABR, specifically the variation Global Best PSO (*gbest* PSO) with Inertia Weight proposed by Shi and Eberhart.

A PSO algorithm maintains a swarm of particles, where each particle represents a potential solution. Let $\vec{u}_i(t)$ denote the position of particle i in the search space at time step t; unless otherwise stated, t denotes discrete time steps. The position of the particle is changed by adding a velocity, $\vec{v}_i(t)$, to the current position, i. e.

$$\vec{u}_i(t+1) = \vec{u}_i(t) + \vec{v}_i(t+1)$$

For *gbest* PSO, the velocity of a particle i is calculated as:

$$v_{id}(t+1) = \omega v_{id}(t) + k_1 r_{1_d}(t)[\mu_{id}(t) - u_{id}(t)] + k_2 r_{2_d}(t)[\hat{\mu}_d(t) - u_{id}(t)]$$

where $v_{id}(t)$ is the velocity of particle i in dimension $d = 1, ..., n_x$ at time step t, $u_{id}(t)$ is the position of particle i in dimension d at time step t, k_1 and k_2 are positive acceleration constants used to scale the contribution of the cognitive and social components respectively, $r_{1_d}(t)$, $r_{2_d}(t) \sim U(0,1)$ are random values in the range [0,1], sampled from a uniform distribution. The inertia weight, ω, controls the momentum of the particle by weighting the previous velocity, basically controlling how much memory of the previous flight direction will influence the new velocity. The personal best position, $\vec{\mu}_i$, associated with particle i is the best position the particle has visited since the first time step. The global best position, $\vec{\hat{\mu}}_d$ is the best position the entire swarm has visited since the first time step.

So the current position of particle i can be expressed as

$$\vec{u}_i(t+1) = \vec{u}_i(t) + \omega\vec{v}_i(t) + k_1\vec{r_1}(t). *^1 [\vec{\mu}_i(t) - \vec{u}_i] + k_2\vec{r_2}(t). * [\vec{\hat{\mu}}(t) - \vec{u}_i(t)]$$

Considering minimization problems, the personal best position at the next time step, t+1, is calculated as

$$\vec{\mu}_i(t+1) = \begin{cases} \vec{\mu}_i(t) & if\ f(\vec{u}_i(t+1)) \geq f(\vec{\mu}_i(t)) \\ \vec{u}_i(t+1) & if\ f(\vec{u}_i(t+1)) < f(\vec{\mu}_i(t)) \end{cases}$$

where $f : \mathbb{R}^{n_x} \to \mathbb{R}$ is the fitness function. The fitness function measures how close the corresponding solution is to the optimum, i.e. the fitness function quantifies the performance, or quality, of a particle (or solution).

The global best position $\vec{\hat{\mu}}(t)$, at time step t, is defined as

$$\hat{\vec{\mu}}(t) \in \{\vec{\mu}_1(t), ..., \vec{\mu}_{n_s}(t)\} \,|f(\vec{\hat{\mu}}(t)) = min\,\{f(\vec{\mu}_1(t)), ..., f(\vec{\mu}_{n_s}(t))\}$$

where n_s is the total number of particles in the swarm. $\vec{\hat{\mu}}(t)$ is the best position discovered by any of the particles so far.

Using the SSABR, we will define one type of agents, the particles, with three states: its coordinates in a n_x dimensional environment $\vec{u}_i(t)$, its velocity $\vec{v}_i(t)$, and the best position the particle has visited since the first time step $\vec{\mu}_i$ (table A.5). The particles also have the parameters k_1, k_2, ω explained at table A.6.

Applying SSABR will lead to the following

Particles

[1] element-wise product

$j = 1$, particle agent type
$\vec{x}^1_{i_1} \in \mathbb{R}^3 \; i_1 = 1, ..., n_1$

$$\vec{x}^1_{i_1} = \begin{pmatrix} \vec{u}_i(t) \\ \vec{v}_i(t) \\ \vec{\mu}_i(t) \end{pmatrix}$$

$X^1(t) = \left(\vec{x}^1_1 \; ... \; \vec{x}^1_{n_1}\right); \; X^1(t) \in M_{3n_x \times n_1}(\mathbb{R})$

We will define some partitions $(x^1_1(t), x^1_2(t), x^1_3(t))$ from $X^1(t)$ for convenience, that allows to select each state for all the agents in a separate way.

$$X^1(t) = \begin{pmatrix} x^1_1(t) \\ x^1_2(t) \\ x^1_3(t) \end{pmatrix}; \; x^1_1(t), \; ..., \; x^1_3(t) \in M_{n_x \times n_1}(\mathbb{R})$$

$\vec{p}^1_{i_1} \in \mathbb{R}^3$

$$\vec{p}^1_{i_1} = \begin{pmatrix} k_1 \\ k_2 \\ \omega \end{pmatrix}$$

$P^1 = \left(\vec{p}^1_1 \; ... \; \vec{p}^1_{n_1}\right); \; P^1 \in M_{3 \times n_1}(\mathbb{R})$
$\vec{p}^1_1 = \vec{p}^1_2 = \; ... \; = \vec{p}^1_{n_1} \equiv \vec{p}^1$ All agents share the same set of parameters

State	Description
$\vec{u}_i(t)$	coordinates of particle i in a n_x dimensional environment
$\vec{v}_i(t)$	velocity of particle i
$\vec{\mu}_i(t)$	the best position the particle i has visited since the first time step

Table A.5: Particle states

Parameter	Description
k_1	acceleration constant of cognitive component
k_2	acceleration constant of social component
ω	inertia weight

Table A.6: Parameters

The environment
The environment is a n_x dimensional world, constrained or unconstrained depending on the problem to solve.

Dynamic methods This optimization algorithm can be described with three methods: the update of the best position each particle has visited, the update of velocity of each particle and the update of the position of each particle. These three methods are iteratively repeated until a stopping condition is satisfied.

Step 1:
$Z^1 = X^1(t) \quad c_1 = \bar{c_1} = n_1$ The number of agents does not change over time.

Step 2:

$$Z^1 = \begin{pmatrix} z_1^1 \\ z_2^1 \\ z_3^1 \end{pmatrix} \; \bar{Z}^1 = \begin{pmatrix} \bar{z}_1^1 \\ \bar{z}_2^1 \\ \bar{z}_3^1 \end{pmatrix}$$

$z_1^1,\ ...,\ z_3^1 \in M_{n_x \times c_1}(\mathbb{R});\ \bar{z}_1^1,\ ...,\ \bar{z}_3^1 \in M_{n_x \times \bar{c}_1}(\mathbb{R})$

Partitions in Z^1 and $\bar{Z}^1$ for convenience, as in the definitions of particle agent states.

$$P^1 = \begin{pmatrix} p_1^1 \\ p_2^1 \\ p_3^1 \end{pmatrix}$$

$p_1^1,\ ...,\ p_3^1 \in M_{1 \times c_1}(\mathbb{R})$ Partitions for convenience

Method 1: At each time step, for each agent, the best personal position is calculated.

$\left(\bar{z}_3^1 \right) = F_{i_1,1}^1(z_1^1,\ z_3^1)$

Method 2: At each time step, for each agent, the velocity of the agent is updated.

$\left(\bar{z}_2^1 \right) = F_{i_1,2}^1(z_1^1,\ z_2^1,\ z_3^1,\ P^1)$

Method 3: At each time step, for each agent, the position of the agent is updated.

$\left(\bar{z}_1^1 \right) = F_{i_1,3}^1(z_1^1,\ z_2^1,\ z_3^1,\ P^1)$

Step 3:
$X^1(t+1) = \bar{Z}^1$

Aggregated variables

In this algorithm two aggregated variables are calculated: $\vec{y}_1$ and y_2.

At each time step, the global best position the entire swarm has visited is calculated ($\vec{y}_1(t) = \vec{\hat{\mu}}(t)$)

$\vec{y}_1(t) = g_1(x_3^1(t))$

The value of the objective function on that point is also calculated.

$y_2(t) = g_2(g_1(x_3^1(t))) = g(x_3^1(t))$

$g_2 \equiv f$ (fitness function)

Appendix B

Application of GPGPU in Optimization

Summary: This section deals with reducing the computational time of optimization processes that use the multi-start optimization technique, by using the power of GPGPU.

B.1 The multi-start optimization method

This work was focused on applying the GPU parallel programming to the multi-start optimization technique. The idea is to launch several optimization processes from different starting points to avoid the algorithm getting stuck in local minima. The main problem of local search algorithms is that they may obtain locally optimal solutions that can be very far (in terms of the objective function value) from the global optimal solution. There are different options to solve this problem of convergence. One of the simplest is to use the methods called multi-start [80].

In the multi-start method, a phase of generation of initial solutions is followed by other of improvement of solutions. The algorithm returns as output the best solution of the objective function value found. The basic multi-start search is characterized by randomly generated initial solutions.

The optimization algorithms used in this part are the Random Optimization (RO) algorithms, a family of numerical optimization methods that do not require using the gradient of the problem and they can therefore be used in functions that are not continuous or differentiable. These methods are also known as direct search algorithms or black box. The name, random optimization, is attributed to Matyas [82], who made a first presentation of the RO with basic mathematical analysis. The RO works iteratively moving to better positions in the search space, which are obtained using a normal distribution around the current position. One of the most significant features of this algorithm is that it ensures convergence to a global minimum with probability 1 in a compact set when the number of iterations tends to infinity. In 1981, Solis and Wets [118] proposed MROM method (Modified Random Optimization Method), a version of the Matyas' algorithm that ensures convergence to the global minimum in a smaller number of iterations. These RO methods are local search algorithms, that is, they consist of an iterative process that begins in a solution and they look in the neighbourhood for a better solution; if found, they replace the current solution by the new one and the process continues until the current solution cannot be improved.

The description of these two algorithms is detailed below. Let $f(\vec{x})$ be the objective function, X the search region, and k the step iteration counter.

Random Optimization Method

1. Select an initial point $\vec{x}(0)$. Let $k = 0$ and M be the total number of iterations.

2. Generate a Gaussian random vector $\vec{\xi}(k)$. If $\vec{x}(k) + \vec{\xi}(k) \in X$, go to step 3. If not, go to step 4.

3. If $f(\vec{x}(k) + \vec{\xi}(k)) < f(\vec{x}(k))$, then $\vec{x}(k+1) = \vec{x}(k) + \vec{\xi}(k)$. If $f(\vec{x}(k) + \vec{\xi}(k)) \geq f(\vec{x}(k))$, then $\vec{x}(k+1) = \vec{x}(k)$.

4. If $k = M$, stop the process. If $k < M$, then k=k+*1* and return to step 2.

Modified Random Optimization Method The only difference from the previous algorithm is step 3:

3.

i If $f(\vec{x}(k)+\vec{\xi}(k)) < f(\vec{x}(k))$, then $\vec{x}(k+1) = \vec{x}(k)+\vec{\xi}(k)$ and $\vec{b}(k+1) = 0.4\vec{\xi}(k) + 0.2\vec{b}(k)$.

ii If $f(\vec{x}(k) + \vec{\xi}(k)) \geq f(\vec{x}(k))$ and $f(\vec{x}(k) - \vec{\xi}(k)) < f(\vec{x}(k))$, then $\vec{x}(k+1) = \vec{x}(k) - \vec{\xi}(k)$ and $\vec{b}(k+1) = \vec{b}(k) - 0.4\vec{\xi}(k)$.

Otherwise, $\vec{x}(k+1) = \vec{x}(k)$ and $\vec{b}(k+1) = 0.5\vec{b}(k)$.

where $\vec{b}$ is the center of the Gaussian random vector $\vec{\xi}$, and $\vec{b}(0) = 0$.

ROM and MROM algorithms were coded in CUDA C and ANSI C using the Visual Studio development environment. The initial solutions were generated from a uniform grid solution space of 49 points, and a set of 10 variances for the optimization methods MROM and ROM were generated, that is, 490 different initial conditions. These 490 optimizations were executed sequentially and in parallel. The objective functions were function

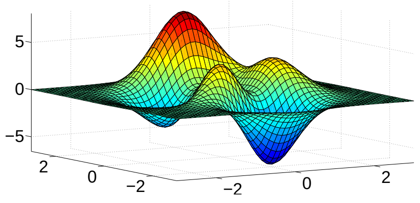


Figure B.1: Peaks function representation.

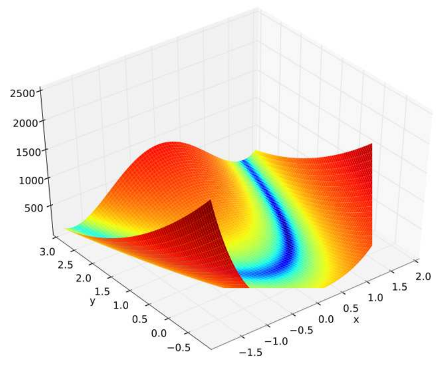


Figure B.2: Two-dimensional Rosenbrock function representation.

peaks from MATLAB (equation B.1, figure B.1) and the Benchmark function *Rosenbrock* (equation B.2, figure B.2) with dimension 10 (n_x=10).

$$\begin{aligned} f(x,y) &= 3(1-x)^2\exp(-x^2-(y+1)^2)-10(x/5-x^3-y^5)\exp(-x^2-y^2) \\ &-1/3\exp(-(x+1)^2-y^2) \end{aligned} \tag{B.1}$$

with $f^* = -6.5511$ in the point (0.2283,-1.6255).

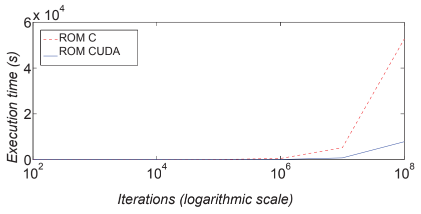
$$f(x) = \sum_{j=1}^{n_x-1}\left[(1-x_j)^2 + 100(x_{j+1}-x_j^2)^2\right] \tag{B.2}$$

with $x_j \in [-2.048, 2.048]$ and $f^* = 0.0$, and n_x the function dimension.

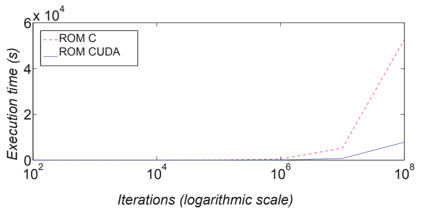
To analyze the execution times of the implementations, the serialized and parallel algorithms were executed, optimizing the objective functions described above, and varying the number of iterations performed by each algorithm, from 10^2 to 10^8 iterations.

The results of execution times can be seen in figures B.3 and B.4, and acceleration factor achieved in each implementation in the tables B.1 and B.2.

In tables B.1 and B.2 it can be observed that the MROM acceleration factor obtained for the same objective function is smaller than the obtained with the ROM algorithm, because the MROM method performs more func-

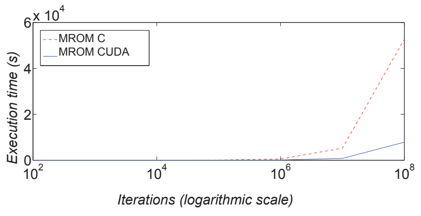


(a) Peaks function

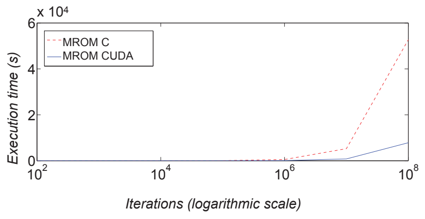


(b) Rosenbrock function

Figure B.3: ROM algorithm execution time.



(a) Peaks function



(b) Rosenbrock function

Figure B.4: MROM algorithm execution time.

tion evaluations. Similarly, evaluating a more complex function (Rosenbrock versus Peaks) supposes less acceleration.

The true efficiency of parallelized code over GPUs is achieved with thousands of threads (in this example 490 threads have been used). To achieve greater efficiency and therefore greater acceleration, memory accesses must be optimized. Global memory is much slower than shared memory, as it

Table B.1: Acceleration factor. Peaks function.

Function Peaks	
Optimization method	**Acceleration factor**
ROM	7x
MROM	5.5x

Table B.2: Acceleration factor. Rosenbrock function.

Function Rosenbrock	
Optimization method	**Acceleration factor**
ROM	5x
MROM	2.5x

is located outside the chip, and therefore it is pretty important to perform the requests efficiently. In the example in this work, we have only worked with global memory. Furthermore, the maximum efficiency is achieved with little divergence, that is, when the code has less conditional statements. The acceleration factor achievable ranges from 3x to 350x depending on the type of application and how the code is optimized.

This work was presented during the conference *XXXIII Jornadas de Automática* that was held in Vigo (Spain) in September of 2012 [103].

Bibliography

Y así, del mucho leer y del poco dormir,
se le secó el celebro de manera que vino
a perder el juicio.

Miguel de Cervantes Saavedra

[1] B. G. Aaby, K. S. Perumalla, and S. K. Seal. Efficient simulation of agent-based models on multi-GPU and multi-core clusters. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, SIMUTools '10, pages 29:1–29:10, ICST, Brussels, Belgium, Belgium, 2010. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[2] Accelereyes. Accelereyes blog. `http://blog.accelereyes.com/blog/2012/12/12/exciting-updates-from-accelereyes/` (last visited September 2013).

[3] Accelereyes. Jacket. `http://www.accelereyes.com/download_jacket` (last visited November 2012), 2012.

[4] B. Al-kazemi and C. Mohan. Training feedforward neural networks using multi-phase particle swarm optimization. In *Neural Information Processing, 2002. ICONIP '02. Proceedings of the 9th International Conference on*, volume 5, pages 2615–2619 vol.5, 2002.

[5] K. Anderson and S. Sheffield. Water Reuse. `http://www.sheffy6marketing.com/` (last visited April 2013).

[6] AndroMeta LLC. AndroMeta. `http://g6g-softwaredirectory.com/bio/cross-omics/agent-based/20451AndroMeta.php` (last visited April 2013).

[7] AnyLogic. Multimethod simulation software. `http://www.anylogic.com` (last visited August 2013).

[8] P. Arpaia, F. Donnarumma, S. Manfredi, and C. Manna. Model predictive control strategy based on differential discrete particle swarm optimization. In *Environmental Energy and Structural Monitoring Systems (EESMS), 2010 IEEE Workshop on*, pages 70–73, 2010.

[9] R. Axelrod. Advancing the art of simulation in the social sciences. *Complexity*, 3(2):16–22, Nov. 1997.

[10] R. Axelrod and L. Tesfatsion. A guide for newcomers to agent-based modeling in the social sciences. In L. Tesfatsion and K. L. Judd, editors, *Handbook of Computational Economics, Vol. 2: Agent-Based Computational Economics*. North Holland, 2006.

[11] R. L. Axtell. Why agents? on the varied motivations for agent computing in the social sciences. In C. M. Macal and D. Sallach, editors, *Proceedings of the Workshop on Agent Simulation: Applications, Models, and Tools*. Argonne National Laboratory, 2000.

[12] C. Blum and D. Merkle. *Swarm Intelligence: Introduction and Applications*. Natural Computing Series. Springer-Verlag Berlin Heidelberg, 2008.

[13] E. Bonabeau. Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences*, 99(90003):7280–7, 2002.

[14] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: from natural to artificial systems*. Oxford University Press, Inc., New York, NY, USA, 1999.

[15] S. Cagnoni and M. Mordonini. Particle swarm optimization and image analysis. In J. R. Rabuñal, J. Dorado, and A. Pazos, editors, *Encyclopedia of Artificial Intelligence*, pages 1303–1309. IGI Global, 2009.

[16] A. Carlisle and G. Dozier. Adapting particle swarm optimisation to dynamic environments. In *International Conference on Artificial Intelligence*, pages 429–434. CSREA Press, 2000.

[17] S. Chandrasekaran, R. K. Suresh, S. G. Ponnambalam, and N. Vijayakumar. An application of particle swarm optimization algorithm to permutation flowshop scheduling problems to minimize makespan, total flowtime and completion time variance. In *CASE*, pages 513–518. IEEE, 2006.

[18] N. Collier. RePast: An extensible framework for agent simulation. `http://repast.sourceforge.net/` (last visited August 2013), 2003.

[19] S. Cook. *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs.* Applications of GPU Computing Series. Elsevier Science, 2012.

[20] E. Dens, K. Bernaerts, A. Standaert, and J. V. Impe. Cell division theory and individual-based modeling of microbial lag: Part i. the theory of cell division. *International Journal of Food Microbiology*, 101(3):303 – 318, 2005.

[21] E. Dens, K. Bernaerts, A. Standaert, J.-U. Kreft, and J. V. Impe. Cell division theory and individual-based modeling of microbial lag: Part ii. modeling lag phenomena induced by temperature shifts. *International Journal of Food Microbiology*, 101(3):319 – 332, 2005.

[22] A. Drogoul, D. Vanbergue, and T. Meurisse. *Multi-Agent Based Simulation: Where are the Agents?*, pages 43–49. Springer, 2003.

[23] R. C. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micromachine and Human Science, Nagoya, Japan.*, pages 39–43, 1995.

[24] R. C. Eberhart and J. Kennedy. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks, Piscataway, NJ.*, pages pp. 1942–1948, 1995.

[25] B. Edmonds. The use of models - making MABS more informative. In *Proceedings of the Second International Workshop on Multi-Agent-Based Simulation-Revised and Additional Papers*, MABS '00, pages 15–32, London, UK, UK, 2001. Springer-Verlag.

[26] A. Engelbrecht. *Fundamentals of Computational Swarm Intelligence.* Wiley, 2005.

[27] J. M. Epstein. Agent-based computational models and generative social science. *Complexity*, 4:41–60, 1999.

[28] J. M. Epstein and R. Axtell. *Growing artificial societies: social science from the bottom up.* Brookings Institution Press, Washington, DC, 1996.

[29] J. D. Farmer and D. Foley. The economy needs agent-based modelling. *Nature*, 460(7256):685–6, Aug. 2009.

[30] J. Ferrer, J. Vidal, C. Prats, J. Valls, E. Herreros, D. López, A. Giró, and D. Gargallo. Individual-based model and simulation of plasmodium falciparum infected erythrocyte in vitro cultures. *Journal of Theoretical Biology*, 248(3):448–59, 2007.

[31] FLAME. Flexible Large-scale Agent Modelling Environment. `http://www.flame.ac.uk` (last visited October 2013).

[32] S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, ECAI '96, pages 21–35, London, UK, UK, 1997. Springer-Verlag.

[33] J. M. Galán, L. R. Izquierdo, S. S. Izquierdo, J. I. Santos, R. del Olmo, A. López-Paredes, and B. Edmonds. Errors and artefacts in agent-based modelling. *Journal of Artificial Societies and Social Simulation*, 12(1):1, 2009.

[34] G. N. Gilbert. *Agent-based models.* Quantitative applications in the social sciences. Sage, 2008.

[35] M. Ginovart and J. Cañadas. INDISIM-YEAST: an individual-based simulator on a website for experimenting and investigating diverse dynamics of yeast populations in liquid media. *Journal of Industrial Microbiology & Biotechnology*, 35(11):1359–1366, 2008.

[36] M. Ginovart, D. López, and A. Gras. Individual-based modelling of microbial activity to study mineralization of C and N and nitrification process in soil. *Nonlinear Analysis: Real World Applications*, 6(4):773 – 795, 2005.

[37] M. Ginovart, D. López, and M. Silbert. Simulation modelling of bacterial growth in yoghurt. *International Journal of Food Microbiology*, 73(2-3):415–25, Mar. 2002.

[38] M. Ginovart, D. López, and J. Valls. INDISIM, an individual-based discrete simulation model to study bacterial cultures. *Journal of Theoretical Biology*, 214(2):305–319, Jan. 2002.

[39] M. Ginovart, D. López, J. Valls, and M. Silbert. Individual based simulations of bacterial growth on agar plates. *Physica A: Statistical Mechanics and its Applications*, 305(3):604–618, 2002.

[40] M. Ginovart and C. Prats. A bacterial individual-based virtual bioreactor to test handling protocols in a NetLogo platform. In *MATHMOD 2012 - 7th Vienna International Conference on Mathematical Modelling*, pages 647–652, 2012.

[41] V. Grimm. Ten years of individual-based modelling in ecology: what have we learned and what could we learn in the future? *Ecological Modelling*, 115(2-3):129–148, Feb. 1999.

[42] V. Grimm, U. Berger, F. Bastiansen, S. Eliassen, V. Ginot, J. Giske, J. Goss-Custard, T. Grand, S. K. Heinz, G. Huse, A. Huth, J. U. Jepsen, C. Jorgensen, W. M. Mooij, B. Muller, G. Pe'er, C. Piou, S. F. Railsback, A. M. Robbins, M. M. Robbins, E. Rossmanith, N. Ruger, E. Strand, S. Souissi, R. A. Stillman, R. Vabo, U. Visser, and D. L. Deangelis. A standard protocol for describing individual-based and agent-based models. *Ecological Modelling*, 198:115–126, 2006.

[43] V. Grimm, U. Berger, D. L. DeAngelis, J. G. Polhill, J. Giske, and S. F. Railsback. The ODD protocol: A review and first update. *Ecological Modelling*, 221(23):2760–2768, Nov. 2010.

[44] W. Gujer. Microscopic versus macroscopic biomass models in activated sludge systems. *Water Science & Technology (IWA Publising)*, 45(6):1–11, 2002.

[45] R. A. Hanneman and M. Riddle. Introduction to social network methods. Riverside, CA: University of California, Riverside (published in digital form at http://faculty.ucr.edu/ hanneman/), 2005.

[46] M. Hare and P. Deadman. Further towards a taxonomy of agent-based simulation models in environmental management. *Mathematics and Computers in Simulation*, 64(1):25–40, Jan. 2004.

[47] I. Harvey, E. A. Di Paolo, R. Wood, M. Quinn, and E. Tuci. Evolutionary robotics: A new scientific tool for studying cognition. *Artificial Life*, 11(1-2):79–98, 2005.

[48] I. Harvey, P. Husbands, and D. Cliff. Issues in evolutionary robotics. In J. Meyer, H. Roitblat, and S. Wilson, editors, *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 364–373. MIT Press, Cambridge, MA, 1993.

[49] S. Hassan, J. Arroyo, J. M. Galán, L. Antunes, and J. Pavón. Asking the oracle: Introducing forecasting principles into agent-based modelling. *Journal of Artificial Societies and Social Simulation*, 16(3):13, 2013.

[50] D. Helbing, editor. *Social Self-Organization: Agent-Based Simulations and Experiments to Study Emergent Social Behavior*. Understanding Complex Systems. Springer, Berlin, 2012.

[51] F. Hellweger and E. Kianirad. Individual-based modeling of phytoplankton: evaluating approaches for applying the cell quota model. *Journal of Theoretical Biology*, 249(3):554–65, 2007.

[52] F. L. Hellweger and V. Bucci. A bunch of tiny individuals -individual-based modeling for microbes. *Ecological Modelling*, 220(1):8–22, 2009.

[53] M. Henze, W. Gujer, and T. Mino, editors. *Activated sludge models ASM1, ASM2, ASM2d and ASM3*, volume No. 9. IWA Publishing, 2000.

[54] A. J. Heppenstall, A. T. Crooks, L. M. See, and M. Batty, editors. *Agent-Based Models of Geographical Systems*. Springer Netherlands, Dordrecht, 2012.

[55] A. G. Hernández-Díaz, L. V. Santana-Quintero, C. Coello Coello, R. Caballero, and J. Molina. A new proposal for multi-objective optimization using differential evolution and rough sets theory. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, GECCO '06, pages 675–682, New York, NY, USA, 2006. ACM.

[56] F. Hinkelmann, D. Murrugarra, A. S. Jarrah, and R. C. Laubenbacher. A mathematical framework for agent based models of complex biological networks. *Computing Research repository*, abs/1006.0408, 2010.

[57] S. Hoshino. On Davies, Swann and Campey minimisation process. *The Computer Journal*, 14(4):426–427, Nov. 1971.

[58] X. Hu and R. Eberhart. Solving constrained nonlinear optimization problems with particle swarm optimization. In *6th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2002)*, pages 203–206, 2002.

[59] S. R. Karnik, A. B. Raju, and M. S. Raviprakasha. Robust design of power system stabilizer using taguchi technique and particle swarm optimization. In *Emerging Trends in Engineering and Technology (ICETET), 2009 2nd International Conference on*, pages 515–520, 2009.

[60] F. Kawai, H. Ito, C. Nakazawa, T. Matsui, Y. Fukuyama, R. Suzuki, and E. Aiyoshi. Automatic tuning for model predictive control: Can particle swarm optimization find a better parameter? In *Intelligent Control, 2007. ISIC 2007. IEEE 22nd International Symposium on*, pages 646–651, 2007.

[61] J. Kennedy and R. C. Eberhart. *Swarm intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[62] D. Kirk and W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Elsevier Science, 2012.

[63] F. Klügl, R. Herrler, and C. Oechslein. From simulated to real environments: How to use SeSAm for software development. In *Multiagent System Technologies - 1st German Conferences MATES, (LNAI 2831*, pages 13–24. Springer, 2003.

[64] F. Klügl. A validation methodology for agent-based simulations. In *Proceedings of the 2008 ACM symposium on Applied computing*, SAC '08, pages 39–43, New York, NY, USA, 2008. ACM.

[65] J.-U. Kreft, G. Booth, and J. W. T. Wimpenny. BacSim, a simulator for individual-based modelling of bacterial colony growth. *Microbiology*, 144(12):3275–3287, Dec. 1998.

[66] J.-U. Kreft, C. Picioreanu, J. W. T. Wimpenny, and M. C. M. van Loosdrecht. Individual-based modelling of biofilms. *Microbiology*, 147(11):2897–2912, 2001.

[67] E. Laskari, K. Parsopoulos, and M. Vrahatis. Particle swarm optimization for minimax problems. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, volume 2, pages 1576–1581, 2002.

[68] R. Leombruni and M. Richiardi. Why are economists sceptical about agent-based simulations? *Physica A: Statistical Mechanics and its Applications*, 355(1):103–109, Sept. 2005.

[69] X. Li, P. Tian, and M. Kong. A novel particle swarm optimization for constrained optimization problems. In S. Zhang and R. Jarvis, editors, *AI 2005: Advances in Artificial Intelligence*, volume 3809 of *Lecture Notes in Computer Science*, pages 1305–1310. Springer Berlin Heidelberg, 2005.

[70] C.-J. Liao, C.-T. Tseng, and P. Luarn. A discrete version of particle swarm optimization for flowshop scheduling problems. *Computers & Operations Research*, 34(10):3099 – 3111, 2007.

[71] J. Lowden. Evolution of the NVIDIA GPU architecture. Advanced Computer Architecture `http://meseec.ce.rit.edu/722-projects/fall2012/1-4.pdf` (last visited October 2013), 2012.

[72] M. Luck, P. McBurney, O. Shehory, and S. Willmott. *Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing).* AgentLink, 2005.

[73] S. Luke, G. C. Balan, L. Panait, C. Cioffi-Revilla, and S. Paus. MASON: A Java multi-agent simulation library. In C. M. Macal, M. North, and D. Sallach, editors, *Proceedings of Agent 2003, Conference on Challenges in Social Simulation.* Argonne National Laboratory, 2003.

[74] M. Lysenko and R. M. D'Souza. A framework for megascale agent based model simulations on graphics processing units. *Journal of Artificial Societies and Social Simulation*, 11(4):10, 2008.

[75] C. M. Macal and M. J. North. Tutorial on agent-based modeling and simulation part 2: How to model with agents. In *Winter Simulation Conference, 2006. WSC 06. Proceedings of the*, pages 73–83, Dec. 2006.

[76] C. M. Macal and M. J. North. Introductory tutorial: agent-based modeling and simulation. In S. Jain, R. R. C. Jr., J. Himmelspach, K. P. White, and M. C. Fu, editors, *Winter Simulation Conference*, pages 1456–1469. WSC, 2011.

[77] V. Mangat. Survey on particle swarm optimization based clustering analysis. In *Proceedings of the 2012 international conference on Swarm and Evolutionary Computation*, SIDE'12, pages 301–309, Berlin, Heidelberg, 2012. Springer-Verlag.

[78] S. Mardani, A. Mirbagheri, M. Amin, and M. Ghasemian. Determination of biokinetic coefficients for activated sludge processes on municipal wastewater. *Iranian Journal of Environmental Health Science & Engineering*, 8(1), 2011.

[79] Y. Marinakis, G. Iordanidou, and M. Marinaki. Particle swarm optimization for the vehicle routing problem with stochastic demands. *Applied Soft Computing*, 13(4):1693–1704, 2013.

[80] R. Martí and J. M. Moreno-Vega. Métodos multi-arranque. *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial*, 2003.

[81] MATLAB. *version 7.10.0 (R2010b)*. The MathWorks Inc., Natick, Massachusetts, 2010.

[82] J. Matyas. Random optimization. *Automation and Remote Control*, 26(2):244–251, 1965.

[83] I. Metcalf & Eddy, G. Tchobanoglous, F. Burton, and H. Stensel. *Wastewater Engineering: Treatment and Reuse*. McGraw-Hill Series in water resources and environmental engineering. McGraw-Hill Education, 2002.

[84] J. Miller and S. Page. *Complex Adaptive Systems: An Introduction to Computational Models of Social Life: An Introduction to Computational Models of Social Life*. Princeton Studies in Complexity. Princeton University Press, 2009.

[85] M. M. Millonas. Swarms, phase transitions and collective intelligence. In C. Langton, editor, *Artificial Life III*. Addison-Wesley, 1994.

[86] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The swarm simulation system: A toolkit for building multi-agent simulations. Santa Fe Institute working paper 96-06-042. Swarm available at `http://www.swarm.org` (last visited August 2013), 1996.

[87] J. Monod. The growth of bacterial cultures. *Annual Review of Microbiology*, 3:371–394, 1949.

[88] R. Moreno. *Estimación de estados y control predictivo del proceso de fangos activados.* PhD thesis, Universitat Autònoma de Barcelona, 1994.

[89] S. Moss. Control metaphors in the modelling of economic learning and decision-making behavior. *Computational Economics*, 8, 1998.

[90] L. Mussi, F. Daolio, and S. Cagnoni. Evaluation of parallel particle swarm optimization algorithms within the CUDATM architecture. *Journal of Information Science*, 181(20):4642–4657, Oct. 2011.

[91] NVIDIA. NVIDIA CUDA C programming guide, version 4.2.

[92] NVIDIA. NVIDIA CUDA C programming best practices guide, version 4.2, 2012.

[93] C. Ocampo-Martinez. Concluding remarks. In *Model Predictive Control of Wastewater Systems*, Advances in Industrial Control, pages 195–200. Springer London, 2010.

[94] G. Pan, Q. Dou, , and X. Liu. Performance of two improved particle swarm optimization in dynamic optimization environments. In *Proceedings of the 6th International Conference on Intelligent Systems Design and Applications*, pages 1024–1028. IEEE Press, 2006.

[95] E. Papageorgiou, K. Parsopoulos, C. Stylios, P. Groumpos, and M. Vrahatis. Fuzzy cognitive maps learning using particle swarm optimization. *Journal of Intelligent Information Systems*, 25(1):95–121, 2005.

[96] K. E. Parsopoulos, K. Skouri, and M. N. Vrahatis. Particle swarm optimization for tackling continuous review inventory models. In M. Giacobini, A. Brabazon, S. Cagnoni, G. D. Caro, R. Drechsler, A. Ekárt, A. Esparcia-Alcázar, M. Farooq, A. Fink, J. McCormack, M. O'Neill, J. Romero, F. Rothlauf, G. Squillero, S. Uyar, and S. Yang, editors, *EvoWorkshops*, volume 4974 of *Lecture Notes in Computer Science*, pages 103–112. Springer, 2008.

[97] K. E. Parsopoulos and M. N. Vrahatis. Recent approaches to global optimization problems through particle swarm optimization. *Natural Computing*, 1(2-3):235–306, 2002.

[98] J. Pavón, A. López-Paredes, and J. Galán. Modelado basado en agentes para el estudio de sistemas complejos. *Novática*, 218:13–18, 2012.

[99] G. U. Y. Pe'er, D. Saltz, and K. Frank. Virtual corridors for conservation management. *Conservation Biology*, 19(6):1997–2003, Dec. 2005.

[100] M. Pereda and J. M. Zamarreño. Agent-based modeling of an activated sludge process in a batch reactor. In *2011 19th Mediterranean Conference on Control & Automation (MED)*, pages 1128–1133, Corfu, June 2011. IEEE.

[101] M. Pereda and J. M. Zamarreño. Modelado basado en agentes. In *Actas de las XXXII Jornadas de Automática*, pages 423–428. Universidad de Sevilla y CEA-IFAC, Septiembre 2011.

[102] M. Pereda and J. M. Zamarreño. An OOP agent-based model for the activated studge process using MATLAB. In S. Omatu, J. F. De Paz Santana, S. R. González, J. M. Molina, A. M. Bernardos, and J. M. C. Rodríguez, editors, *Distributed Computing and Artificial Intelligence*, volume 151 of *Advances in Intelligent and Soft Computing*, pages 241–248. Springer Berlin Heidelberg, 2012.

[103] M. Pereda and J. M. Zamarreño. Optimización multiarranque en paralelo sobre GPU. In R. S. Matías García, editor, *Actas de las XXXIII Jornadas de Automática*, pages 423–428. Universidad de Vigo y CEA-IFAC, Septiembre 2012.

[104] B. Potter, J. Sinclair, and D. Till. *Introduction to Formal Specification and Z (2nd Edition).* Prentice Hall PTR, 1996.

[105] C. Prats. *Individual-based Modelling of Bacterial Cultures in the Study of the Lag Phase.* PhD thesis, Universitat Politècnica de Catalunya. Departament de Física i Enginyeria Nuclear and Escola Superior d'Agricultura de Barcelona, 2008.

[106] S. F. Railsback and V. Grimm. *Agent-Based and Individual-Based Modeling: A Practical Introduction.* Princeton University Press, 2011.

[107] A. Rakitianskaia and A. Engelbrecht. Training feedforward neural networks with dynamic particle swarm optimisation. *Swarm Intelligence*, 6(3):233–270, 2012.

[108] A. Repenning and T. Sumner. Agentsheets: A medium for creating domain-oriented visual languages. IEEE Computer 28: 17-25. `http://AgentSheets.com/` (last visited August 2013), 1995.

[109] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Computer Graphics*, 21(4):25–34, Aug. 1987.

[110] M. Richiardi, R. Leombruni, N. J. Saam, and M. Sonnessa. A common protocol for agent-based social simulation. *Journal of Artificial Societies and Social Simulation*, 9(1):15, 2006.

[111] P. Richmond. FLAME GPU technical report and user guide. technical report CS-11-03. Technical report, University of Sheffield, Department of Computer Science, 2011.

[112] D. L. Russell. *Practical Wastewater Treatment*. John Wiley & Sons, Inc., 2006.

[113] J. Sanders and E. Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Pearson Education, 2010.

[114] T. C. Schelling. Models of segregation. *The American Economic Review*, 59(2):488–493, 1969.

[115] A. J. Schuler. Diversity matters: dynamic simulation of distributed bacterial states in suspended growth biological wastewater treatment systems. *Biotechnology and Bioengineering*, 91:62–74, 2005.

[116] A. J. Schuler, N. Majed, V. Bucci, F. L. Hellweger, Y. Tu, and A. Z. Gu. Is the whole the sum of its parts? agent-based modelling of wastewater treatment systems. *Water Science & Technology (IWA Publising)*, 63(8):1590, 2011.

[117] Y. Shi and R. C. Eberhart. A modified particle swarm optimizer. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 69 – 73, 1998.

[118] F. J. Solis and R. J. B. Wets. Minimization by random search techniques. *Mathematics of Operations Research*, 6(1):19–30, 1981.

[119] A. Talaie khozani, N. Jafarzadeh Haghighi Fard, M. Talaie Khozani, and M. Beheshti. The determination of biokinetic coefficients of crude oil biodegradation using pseudomonas aeruginosa bacteria. *Health & Environment*, 3:111–122, 2010.

[120] Y. Tan and Y. Zhou. Parallel particle swarm optimization algorithm based on graphic processing units. In B. Panigrahi, Y. Shi, and M.-H. Lim, editors, *Handbook of Swarm Intelligence*, volume 8 of *Adaptation, Learning, and Optimization*, pages 133–154. Springer Berlin Heidelberg, 2010.

[121] W. Tang. Accelerating agent-based modeling using graphics processing units. In X. Shi, V. Kindratenko, and C. Yang, editors, *Modern Accelerator Technologies for Geographic Information Science*, pages 113–129. Springer US, 2013.

[122] W. Tang, D. A. Bennett, and S. Wang. A parallel agent-based model of land use opinions. *Journal of Land Use Science*, 6(2-3):121–135, 2011.

[123] A. Taylor. The verification of dynamic simulation models. *Journal of the Operational Research Society*, 34(3):233–242, 1983.

[124] L. Tesfatsion. Agent-based computational economics: Growing economies from the bottom up. *Artifitial Life*, 8(1):55–82, Mar. 2002.

[125] L. Tesfatsion. Agent-Based Computational Economics. Growing Economies from the Bottom Up. `http://www.econ.iastate.edu/tesfatsi/` (last visited March 2013), 2013.

[126] I. Torsun. *Foundations of Intelligent Knowledge-Based Systems*. Library and Information Science. Academic Press Limited, 1995.

[127] F. van den Bergh and A. Engelbrecht. A study of particle swarm optimization particle trajectories. *Information Sciences*, 176(8):937–971, 2006.

[128] U. Wilensky. NetLogo Segregation model. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. `http://ccl.northwestern.edu/netlogo/models/Segregation` (last visited March 2013), 1997.

[129] U. Wilensky. NetLogo. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.`http://ccl.northwestern.edu/netlogo/` (last visited March 2013), 1999.

[130] U. Wilensky. NetLogo Sugarscape 1 Immediate Growback model. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. `http://ccl.northwestern.edu/netlogo/models/Sugarscape1ImmediateGrowback` (last visited March 2013), 2009.

[131] I. Wolfram Research. *Mathematica*. Wolfram Research, Inc., 2010.

[132] D. Xiao, D. Song, L. Peng, and T. Li. Hybrid model predictive control based on modified particle swarm optimization. In *BIC-TA*, pages 385–390. IEEE, 2010.

[133] Y. Zhou and Y. Tan. GPU-based parallel particle swarm optimization. In *Proceedings of the Eleventh conference on Congress on Evolutionary Computation*, CEC'09, pages 1493–1500, Piscataway, NJ, USA, 2009. IEEE Press.

[134] Q. Zou, J. Ji, S. Zhang, M. Shi, and Y. Luo. Model predictive control based on particle swarm optimization of greenhouse climate for saving energy consumption. In *World Automation Congress (WAC), 2010*, pages 123–128, 2010.